Strukturverträgliche Ontologien der Automatisierungstechnik

Von der Fakultät für Maschinenbau
der Technischen Universität Carolo-Wilhelmina zu Braunschweig

zur Erlangung der Würde
eines Doktor-Ingenieurs (Dr.-Ing.)

genehmigte

Dissertation

von Dipl.-Ing. **Lorenz Däubler**

aus Ansbach

Eingereicht am:	6. Juli 2005
Mündliche Prüfung am:	25. November 2005
Berichterstatter:	Univ.-Prof. Dr.-Ing. E. Schnieder
	Univ.-Prof. Dr. rer. nat. M. Polke

2006

Vorwort

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Institut für Verkehrssicherheit und Automatisierungstechnik (ehemals Institut für Regelungs- und Automatisierungstechnik) der TU Braunschweig. An dieser Stelle möchte ich mich bei allen bedanken, die zum Gelingen der Arbeit beigetragen haben.

An erster Stelle gilt mein Dank Herrn Prof. Dr.-Ing. E. Schnieder, dem Leiter des Instituts für Verkehrssicherheit und Automatisierungstechnik der TU Braunschweig, für die Möglichkeit zur Promotion. Neben dem kooperativen und menschlich sehr angenehmen Arbeitsklima waren es vorallem sein unermüdliches Engagement für grundlegende Begriffsbestimmungen und interdisziplinäre, ganzheitliche Sichtweisen, die mir in Erinnerung bleiben werden.

Herrn Prof. Dr. rer. nat. M. Polke, den ich während meiner Mitarbeit im GMA-Fachausschuss 7.21 kennenlernen durfte, danke ich vorallem für die wertvollen Hinweise zu den philosophischen Aspekten des Ontologiebegriffs und für die Übernahme des Ko-Referats. Herrn Prof. Dr.-Ing. H.-J. Franke vom Institut für Konstruktionstechnik danke ich für die Übernahme des Vorsitzes der Prüfungskommission.

Weiterhin möchte ich bei meinen Institutskollegen für die stets offene und entspannte Arbeitsatmosphäre und die Bereitschaft zu den vielen wertvollen Fachgesprächen bedanken. Wichtige fachliche Unterstützung habe ich zusätzlich von den Herren J. Kelsch, X. Jiang, J. Gehrke, D. Püschmann und Th. Morgeneyer erhalten, die ich während meiner Zeit als wissenschaftlicher Mitarbeiter bei der Verfassung ihrer studentischen Arbeiten betreuen konnte. Dafür möchte ich mich an dieser Stelle ganz herzlich bedanken.

Neben den vorhergenannten Personen möchte ich mich natürlich auch bei meiner Familie bedanken. Da sind zunächst meinen Eltern zu nennen, die mir das Maschinenbau-Studium an der TU Braunschweig ermöglicht haben. Am meisten verdanke ich jedoch meiner Frau Nadine und meiner Tochter Moira, die während der Erstellung dieser Dissertation viele Stunden geduldig auf mich verzichtet haben und mir immer wieder auf liebevolle Weise zeigten, welche die wirklich wichtigen und wertvollen Dinge im Leben sind.

Braunschweig, im Juli 2005

Lorenz Däubler

Inhaltsverzeichnis

Kurzfassung

Gegenstand der vorliegenden Dissertation ist ein integriertes Entwurfsverfahren für Automatisierungssysteme, bei dem sowohl die verhaltenbeschreibende als auch die gerätetechnische Sicht gleichermaßen Berücksichtigung finden. Dieses Entwurfsverfahren basiert auf der Erkennung, Überprüfung und Einhaltung von Ähnlichkeiten, die zwischen geräte- und verhaltensbeschreibenden Systemmodellen herrschen. So können die während der Geräte- und Anlagenplanung entstehenden Spezifikationen (Stücklisten, Baugruppenhierarchien, Netzwerke) bis in die aus automatisierungstechnischer Sicht notwendige Detailltiefe in den Verhaltensmodellen konsistent nachgezogen werden.

Auf der Basis prozesstheoretischer Überlegungen werden kombinierte Petrinetz- und Bondgraphen-Modelle als dynamische Beschreibungsmittel gewählt, um das für Automatisierungssysteme typische hybride Verhalten darstellen zu können. Für die Beschreibung gerätetechnischer Strukturen dient das STEP-Produktmodell nach ISO 10303.

Um die Ähnlichkeiten zwischen der verhaltenbeschreibenden und der gerätetechnischen Sicht formal fassen zu können, werden die den Systemmodellen zugrundeliegenden Modellkonzepte in Ontologien überführt und diese dann mit strukturverträglichen Abbildungen, sogenannten Morphismen, aufeinander abgebildet.

Sowohl die Ontologien als auch die über diesen Ontologien definierten Morphismen werden mit Mitteln der OMA (*Object Management Architecture*) in MOF/UML-Modelle und OCL-Spezifikationen übertragen. Diese Spezifikationen sind dann die Implementationsgrundlage einer Reihe von Softwarewerkzeugen, die einen Entwurfsrahmen bilden, mit dem das integrierte Entwurfsverfahren anhand von einfachen Beispielen näher untersucht wird.

Stichwörter: Ontologie, Morphismus, Petrinetz, Bondgraph, STEP, ISO 10303, OMA, Prozesstheorie.

1 Einleitung

Ein Automatisierungssystem ist eine technische Einrichtung, die zu dem einen Zweck entwickelt wird, einen ihr übergeordneten Gesamtprozess zielgerichtet und ohne direkten menschlichen Eingriff ablaufen zu lassen. Um den vielfältigen technischen, wirtschaftlichen und rechtlichen Anforderungen an ein Automatisierungssystem gerecht zu werden, muss deshalb der automatisierte Gesamtprozess gewerkespezifisch und von unterschiedlichen Sichten aus betrachtet und analysiert werden. Wie in [Schnieder 2003] gezeigt wird, sind von den vielen unterschiedlichen Sichten auf den Gesamtprozess diejenigen bezüglich der Funktion, des Verhaltens, des Aufbaus und des Zustands von zentraler Bedeutung. Die einzelnen Sichten können begrifflich folgendermaßen präzisiert werden:

- **Funktion**: Die Funktion einer technischen Einrichtung ist die Aufgabe, die diese Einrichtung erfüllen soll. Die technische Einrichtung ist das Mittel zur Erfüllung dieser Aufgabe, wobei die Funktion an sich lösungs- bzw. realisierungsinvariant ist ([Lauber 1996]). Komplexe Funktionen können in Teil- und Einzelfunktionen dekomponiet werden.

- **Aufbau**: Mit dem Aufbau ist die materiale und räumliche Struktur eines Automatisierungssytems gemeint. Dies beinhaltet zum einen alle Geräte und Bauelemente, deren Konfiguration und Komposition in Baugruppen und alle physikalischen Verknüpfungen und Netzwerke zwischen den Baugruppen und Geräten, zum anderen auch deren topologische Anordnung und Orientierung.

- **Verhalten**: In der verhaltenbeschreibenden Sicht werden die durch äußere Auswirkungen oder autonom hervorgerufenen Änderungen von Prozessgrößen untersucht. Von Interesse sind das dynamische Verhalten (zeitliche Prozessänderungen, Transienten), das stationäre Verhalten (Werteänderungen nach dem Abklingen aller zeitlichen Änderungen), das logische Verhalten (kausale Gesetzmäßigkeiten zwischen Ein- und Ausgangsgröße). Grundlage für diese Betrachtungen sind die in den ingenieurwissenschaftlichen Teildisziplinen erarbeiteten physikalischen Zusammenhänge, die im Allgemeinen durch Differenzialgleichungen beschrieben werden.

- **Zustand**: Der Begriff 'Zustand' ist in vielen Ingenieur- und Informatikwissenschaften weit verbreitet, wird dort jedoch oft unterschiedlich interpretiert. Eine einheitliche, physikalisch motivierte Definition findet sich in [Waloschek 1998] mit Zustand als *'... die Form oder die Art, in der ein Körper, eine Substanz oder ein System in einem bestimmten Moment vorliegt'*. Danach sind Zustände also im Allgemeinen zeitabhängig und können durch verhaltendefinierende Prozessoperatoren

in Folgezustände transformiert werden [1]. Die genannte Definition geht aber über den rein zeitlichen Charakter von Zuständen hinaus, da der Zustand auch die Art bzw. Form eines Systems betrifft und somit einen effektiven Ansatzpunkt für die Formulierung und Spezifikation sowohl von funktionalen, räumlich/strukturellen und verhaltensrelevanten als auch von sicherheitstechnischen, rechtlichen und wirtschaftlichen Anforderungen bietet.

Dieses Sichtenkonzept korreliert weitgehend mit den axiomatisch formulierten Systemthesen in [Schnieder 1993], wobei dort die Sichten 'Funktion' und 'Verhalten' den Systemmerkmalen 'Kausalität' und 'Temporalität' entsprechen.

Es lässt sich nun aber feststellen, dass die eingangs erwähnten Sichten eng miteinander verzahnt sind und nicht isoliert voneinander betrachtet werden können. Jede Systemfunktion beispielsweise muss gerätetechnisch berücksichtigt werden und hat somit Auswirkungen auf den Aufbau eines Automatisierungssytems. Darüber hinaus wirkt sich eine Systemfunktion im Allgemeinen auch auf das Prozessverhalten durch gezielte Verhaltensbeeinflussung aus (Verhalten) und lässt sich in Form von erforderlichen bzw. nicht-erwünschten Zuständen spezifizieren (Zustand). Das Verhalten eines Automatisierungssystems wiederum wird ganz entscheidend von den gerätetechnischen Komponenten des zu automatisierenden Prozesses bestimmt (Aufbau) und ermöglicht bei Anwendung entsprechender Modellierungsmethoden auch die Erzeugung zeitlicher/räumlicher Zustandstrajektorien (Zustand).

Die Systemsichten und ihre wechselseitigen Beziehungen untereinander lassen sich anhand eines Tetraeders anschaulich darstellen, siehe Abbildung 1.1. Die Eckpunkte des Tetraeders werden durch die Sichten gebildet; die Kanten des Tetraeders repräsentieren die Beziehungen zwischen den Sichten.

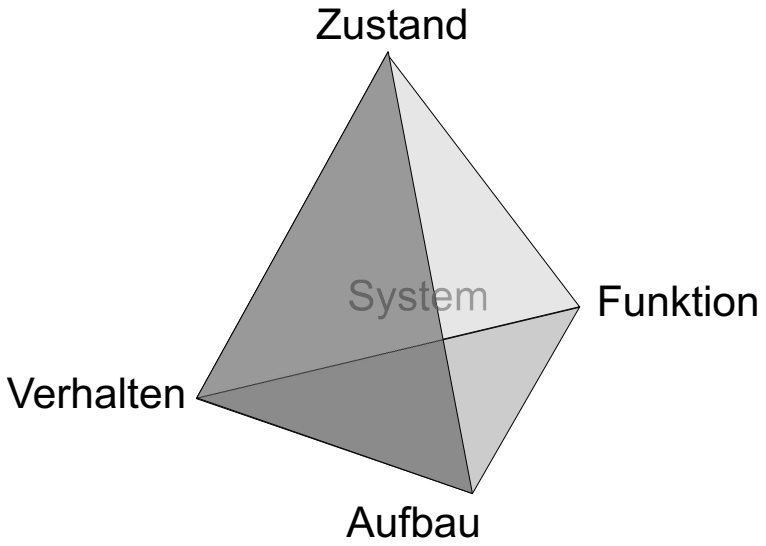


Abbildung 1.1: Systemtetraeder mit unterschiedlichen Sichten

[1] siehe hierzug auch Gleichung (2.77)

Der vorgestellte Systemtetraeder vereint zwar nur in simpler geometrischer Art die vier unterschiedlichen, miteinander verkoppelten Sichten, lässt aber auch ganz anschaulich das idealtypische Ziel des integrierten Entwurfs von Automatisierungssystemen erahnen - nämlich in jeder Sicht die zu Gebote stehende Berücksichtigung aller in den benachbarten Sichten erarbeiteten Anforderungen, die Wiederverwendung der dort erlangten Kenntnisse und die rückwirkende und damit nachhaltige Aktualisierung neuerworbener Informationen über alle Sichtengrenzen hinweg.

Es werden nun im Verlaufe der Einleitung zum einen einige Problemstellungen bei der Integration dieser Sichten im Entwurfsprozess von Automatisierungssystemen dargestellt als auch der in der vorliegenden Dissertation dargestellte Lösungsansatz kurz umrissen. Das Ende der Einleitung bildet ein Überblick über die Gliederung der Arbeit.

1.1 Problemstellung und Lösungsansatz

Der integrierte Entwurfsprozess industriell erzeugter Produkte erhält seit Jahren starke Impulse aus der Produktdatentechnologie. Anfänglich auf den Austausch von CAD-Daten in Form neutraler Datenformate fokusiert, hat sich die Produktdatentechnologie zu einem ISO-zertifizierten, hochkomplexen Standard (STEP - Standard for the Exchange of Product Model Data, ISO 10303) entwickelt, der breite Anwendung insbesondere in der Automobil- und Luftfahrtindustrie findet. Kern der Produktdatentechnologie ist ein komplexes Produktmodell, das als Referenzmodell dient und den Austausch strukturbeschreibender Daten über den gesamten Lebenszyklus eines Produktes in unterschiedlichen Branchen (Luftfahrt, Automobilindustrie, Schiffsbau usw.) ermöglicht. Die unter der Produktdatentechnologie subsumierten Verfahren und Methoden werden dementsprechend auch PDM (Produktdatenmanagement) oder neuerdings PLM (*Product Lifecycle Management*) genannt. Aktuell konzentrieren sich die Forschungs- und Entwicklungstätigkeiten im Bereich der Produktdatentechnologie auf die Integration von verteilten Entwicklungsprozessen (*collaborative engineering*), wie sie immer häufiger bei Hersteller/Zulieferer-Kooperationen zu finden sind.

Diese geräteorientierte Herangehensweise bei der Produktentwicklung birgt jedoch auch Risiken, siehe z.B. [Buchner 1998]: *'Mit der Ausrichtung auf die apparative Gestaltung ... geht somit das Verhalten des Prozesses aus dem Blickfeld verloren'*[2]. Um dem entgegenzuwirken, wurde im Zuge des MechaSTEP-Projektes [Donges 2000] ein Informationsmodell zur Beschreibung simulationsrelevanter Daten entwickelt. MechaSTEP stellt somit einen ersten Ansatz dar, um STEP-konforme Produktinformationen aus der strukturorientierten Sicht (Aufbau) für Untersuchungen des Systemverhaltens (Verhalten) nutzen zu können. Das MechaSTEP-Projekt war zwar auf die Beschreibung sogenannter mechatronischer Systeme, die sich aus elektrisch/elektronischen, hydraulischen oder mechanischen Teilsystemen zusammensetzen können (vgl. [Isermann 1999]), ausgerichtet, jedoch wurden hauptsächlich Datenbestände für Mehrkörpersimulations-

[2] [Buchner 1998], S. 9

systeme (MKS-Systeme) in bestehende PDM-Systeme integriert [Anderl 2002]. Andere für die Automatisierungstechnik notwendige Beschreibungsmittel, insbesondere für informatische Prozesse, wurden nicht untersucht. Desweiteren wurde die vorgeschlagene Integrationsmethodik erschwert, da die in den MKS-Simulationssystemen benötigten Modellstrukturen nicht durchgehend mit Produktstrukturen, wie sie für Konstruktions- oder Produktionsprozesse notwendig sind, korrelieren. Dies liegt daran, dass während der Modellbildung von MKS-Systemen vorhandene gerätetechnische Strukturen durch Vereinfachung, Reduktion oder Abstraktion nicht unverändert bleiben. Diese und auch weitere Gründe (siehe hierzu [Däubler 2004]) sind ausschlaggebend dafür, dass die Verfahren der MechaSTEP-Initiative insbesondere auch im Bereich der Automatisierungstechnik noch wenig Anklang gefunden haben.

Der in dieser Arbeit vorgestellte Ansatz eines integrierten Entwurfsverfahrens für Automatisierungssysteme hat nun zum Ziel, STEP-konforme Produktdaten aus der strukturellen Systemsicht (Aufbau) in der verhaltenbeschreibenden Sicht für Modellbildungszwecke zu nutzen (Verhalten), um dann aufbauend auf dieser Modellbildung im Zustandsraum Analyse- und Simulationsverfahren anzuwenden (Zustand). Die Kopplung der Aufbau- und Verhaltenssicht basiert auf einem neuartigen Verfahren zur Integration strukturverträglicher Ontologien. Dieses ermöglicht die Wiederverwendung von Strukturdaten im Modellbildungsprozess. Spezifiziert und implementiert wird das Verfahren mit Mitteln der OMA (*Object Modeling Architecture*), namentlich UML/OCL, MOF und XMI. Um in der Verhaltenssicht die Modellbildung gemischt kontinuierlich/diskreter Systeme in allen für die Automatisierungstechnik relevanten Teildisziplinen durchführen zu können, kommen als Beschreibungsmittel Bondgraphen für die Modellierung energetischer Prozesse und Petrinetze für die Modellierung informatischer Prozesse zum Einsatz. Den bereits erwähnten Problemen mit unterschiedlichen Abstraktions- und Verfeinerungsgraden in Verhaltens- und Produktmodellen wird durch entsprechende Baugruppenhierarchien in der Aufbausicht begegnet. Die mit Bondgraphen und Petrinetzen erstellten Verhaltensmodelle bieten zusätzlich den Vorteil, dass sie methodisch in Zustandsform überführt und in der Zustandssicht mittels hybrider Automaten nahtlos miteinander gekoppelt werden können.

Das Ergebnis ist eine durchgängige Entwurfsmethode, die es ermöglicht, vorhandene Produktdaten aus der Aufbausicht für die Verhaltensmodellierung wiederzuverwenden.

1.2 Gliederung der Arbeit

In der vorliegenden Dissertation werden sämtliche Begriffe, Konzepte, Methoden, Verfahren, Beschreibungsmittel, Normen und Technologien erläutert, die für das grundlegende Verständnis der erwähnten Entwurfsmethodik notwendig sind.

Auf die am Anfang stehenden einleitenden Erläuterungen folgt das zweite Kapitel, das die theoretischen Grundlagen technischer Prozesse auf der Basis von [Wunsch 2000] behandelt, mit denen es möglich ist, die für die Automatisierungstechnik kennzeichnende

Begriffs- und Konzeptvielfalt formal-algebraisch und einheitlich zu fassen. Im Mittelpunkt stehen vollständige Prozessdarstellungen, temporale und kooperative Wechselwirkungen, die Zustandsdarstellung und hybride Prozesse.

Das dritte Kapitel beinhaltet eine Übersicht über ausgewählte Methoden zur Informationsmodellierung, die die Strukturierung und Formalisierung von Anwendungswissen ermöglichen. Diese Methoden sind Teil der Forschungsgebiete 'Wissensverarbeitung' und 'künstliche Intelligenz'. Im Kontext dieser Arbeit hat sich vor allem der Begriff der Ontologien sowohl theoretsich als auch praktisch als sehr belastbar erwiesen. Ergebnis der dort vorgestellten Untersuchungen ist ein neuartiges Verfahren zur Integration von Ontologien auf der Basis strukturverträglicher Abbildungen, die auch Morphismen genannt werden. Bevor diese Morphismen dann im fünften Kapitel für die Kopplung struktur- und verhaltensbezogener Ontologien herangezogen werden, werden im vierten Kapitel die diesen Ontologien zugrundeliegenden notwendigen Modellkonzepte vorgestellt. Dies sind zum einen Bondgraphen und Petrinetze für die Verhaltensmodellierung, und zum anderen STEP-konforme Produktmodelle nach ISO 10303 - namentlich das Anwendungsprotokoll AP 212 - zur Beschreibung von Gerätestrukturen. Im sechsten Kapitel wird eine umfangreiche Werkzeugumgebung präsentiert, die ausgehend von gegebenen Produktdaten die strukturtreue Modellbildung eines Automatisierungssystems mit Bondgraphen und Petrinetzen ermöglicht. Als nicht-triviales, akademisches Beispiel dient hier ein einfaches Zweitanksystem mit Füllstandsregelung, dessen Verhaltensmodelle anschließend in Zustandsform überführt und simuliert werden.

Den Abschluss bildet das siebte Kapitel mit einer Zusammenfassung des gesamten Inhalts und mit einem Ausblick auf mögliche zukünftige Erweiterungen.

2 Prozesstheoretische Grundlagen

Im folgenden Kapitel werden die theoretischen Grundlagen erarbeitet, um die dynamischen Phänomene, die für das Zusammenspiel von Anlagen und Automatisierungssystemen typisch sind, allgemeingültig einordnen zu können. Diese Grundlagen basieren weitgehend auf der Prozesstheorie nach [Wunsch 2000] und zeichnen sich durch einen hohen Abstraktions- und Formalisierungsgrad aus. Der Schwerpunkt dieser Theorie liegt im Wesentlichen auf der Betrachtung der zeitlichen Änderungen, die die Eigenschaften von Prozessobjekten während eines Prozesses erfahren können - unabhängig davon, welche Ordnungsstrukturen oder Skalierungen diesen Prozesseigenschaften zugrunde liegen. Im Verlauf des Abschnitts 2.6 wird jedoch auf die Bedeutung dieser Ordnungsstrukturen und Skalierungen näher eingegangen.

Das Kapitel ist folgendermaßen gegliedert: Nach der allgemeinen Definition der Begriffe 'System' und 'Prozess' folgen Erläuterungen zur Prozessdarstellung (Vollständigkeit) und zu den temporalen und kooperativen Wechselwirkungen, die innerhalb von Prozessen herrschen können. Auf diesen Wechselwirkungen basieren Prozesskategorien wie 'Determiniertheit', 'Reversibilität' und 'Kausalität'. Besonderes Augenmerk wird dann auf die Prozessdarstellung in Zustandsform gerichtet, da sich aus hieraus weitere systemtheoretische Kenntnisse herleiten lassen. Abschlißend werden die Charakteristika hybrider Prozesse erläutert.

2.1 Systeme und Prozesse

Ziel des folgenden Abschnitts ist es, die Begriffe 'System' und 'Prozess' formal zu fassen. Grundgedanke dieser Formalisierung ist der, dass Systeme im Allgemeinen ein dynamisches Verhalten zeigen und dieses Verhalten durch Prozesse beschrieben werden kann. Prozesse beschreiben damit also, wie sich die Eigenschaften der Objekte eines Systems im Laufe fortschreitender Zeit ändern (vgl. Temporalprinzip in [Schnieder 1993]).

Um diese pragmatische Sicht auf Prozesse und Systeme formalisieren zu können, werden zuerst X als Menge der möglichen Objekteigenschaften und T als Menge der Zeitpunkte, an denen die Eigenschaftsänderungen der Objekte eintreten, eingeführt. Wir nennen T und X auch Grund- oder Trägermengen eines Prozesses.

- X heißt Phasenraum und ist die Menge aller Objekteigenschaften x, auch Phasen oder Phasenpunkte genannt.

- T heißt Zeitbereich oder Zeitraum und ist die Menge aller Zeitpunkte t, zu denen die Änderungen der Objekteigenschaften eintreten.

Der Menge T wird durch die Ordnungsrelation $\leq$ eine Ordnungsstruktur aufgeprägt. Diese Ordnungsstruktur ist wesentlich für die Definition der Kausalität im Sinne der Zeit.

Über die Ordnungsstruktur von T hinaus sollen an dieser Stellen keine weiteren Aussagen zur Struktur der Trägermengen gemacht werden, wobei es gerade dieser Strukturierung ist, die zur Einteilung und Abgrenzung ganzer Wissenschaftsdisziplinen genutzt wird. In der Informatik und Automatisierungstechnik werden oft algebraisch nicht strukturierte Phasenräume (d.h. ohne, dass innere Verknüpfungen wie Addition und Multiplikation auf ihnen definiert sind) betrachtet. Die klassische Regelungstechnik hingegen beschäftigt sich mit geordeneten, algebraisch strukturierten Trägermengen $T \subset \mathbb{R}$, $X \subset \mathbb{R}$ (kontinuierliche Systeme). Die dynamischen Vorgänge in Abtastsysteme wiederum sind Prozesse über $T \subset \mathbb{Z}$, $X \subset \mathbb{R}$. Für eine ausführlichere Darstellung dieser Zusammnehänge wird auf [Schnieder 1993] verwiesen.

Basierend auf diesen Trägermengen kann man nun eine Eigenschaftsänderung durch eine Abbildung ξ von T auf X beschreiben.

- ξ heißt Signal oder auch Verhaltenstrajektorie und ist eine Abbildung der Form

$$\xi : D(\xi) \to X \quad \text{oder} \quad \xi(t) = x \quad \text{mit} \quad x \in X, \quad t \in D(\xi) \subset T \tag{2.1}$$

Bei einem Signal ξ handelt es sich also um eine zweistellige, (rechts)eindeutige Relation:

$$\xi \subset T \times X \tag{2.2}$$

Die Definitionsmenge $D(\xi)$ in (2.1) gewährleistet, dass ξ linkstotal und somit unter der Annahme der Eindeutigkeit eine Abbildung ist [Reinhardt 1991], [Bronstein 1991].

Das kartesische Produkt der Trägermengen in (2.2) heißt auch Ereignisraum und lässt sich folgendermaßen definieren:

- Das kartesische Produkte der Träger T und X

$$T \times X \tag{2.3}$$

heißt Ereignisraum und beinhaltet alle Ereignisse (t, x) mit $t \in T$ und $x \in X$.

Von definitorischer Bedeutung ist nun noch die Menge Ξ^* aller möglichen Signale ξ über T und X. Da ja $\xi \subset T \times X$ ist, muss Ξ^* die Menge aller möglichen Teilmengen von $T \times X$ und damit die Potenzmenge von $T \times X$ sein. Ξ^* wird im weiteren als Signalraum bezeichnet.

- Die Produktmenge

$$\Xi^* = \mathsf{P}(T \times X) = [T, X] \tag{2.4}$$

heißt Signalraum oder Universum aller möglichen Signale ξ über T, X.

Damit sind nun alle Begriffe eingeführt, die zu einer allgemeinen Prozessdefinition notwendig sind:

- Eine Signalmenge

$$\Xi \subset \Xi^* = [T, X] \tag{2.5}$$

mit den Signalen (oder Trajektorien) ξ heißt Prozess über den Trägermengen T und X. (Prozessdarstellung durch Tupel (T, X, Ξ)).

Nach dieser elementaren Definition handelt es sich bei einem Prozess zunächst einmal nur um eine Menge von Signalen, die den zeitlichen Verlauf von Eigenschaften eines beliebigen Objektes beschreiben. Mit dieser Definition können also eine Vielzahl von Phänomenen in Natur, Technik und Gesellschaft, die durch zeitliche Veränderung gekennzeichnet sind, als Prozess aufgefasst werden.

Gleichzeitig wird auch deutlich, dass ein Prozess umso stärker von physikalischen, technischen oder anderen Gesetzmässigkeiten geprägt ist, je stärker er sich von Ξ^* unterscheidet. Mithin repräsentiert der Prozess $\Xi = \Xi^*$ den chaotischen Prozess.

Analog zur Zusammenfassung aller möglichen Signale ξ über T und X in der Menge Ξ^* lassen sich auch alle möglichen Prozesse Ξ in einer Menge $\mathcal{X}^*$ so zusammenfassen, dass gilt: $\Xi \in \mathcal{X}^*$.

- Die Produktmenge

$$\mathcal{X}^* = \mathsf{P}(\Xi^*) \tag{2.6}$$

heißt Menge aller möglichen Prozesse.

Im weiteren Verlauf des Kapitels wird ersichtlich, wie Teilmengen dieser sehr globalen Menge $\mathcal{X}^*$ durch Begriffe wie 'reversibel', 'irreversibel', 'vollständig', oder 'Gedächtnis' charakterisiert und eingegrenzt werden können.

Aufbauend auf dieser Darstellung wird nun abschließend der Systembegriff definiert:

- Das Tupel

$$(T, X, \Xi) \tag{2.7}$$

heißt (dynamisches) System mit dem das Verhalten charakterisierenden Prozess Ξ über den Grund- oder Trägermengen T, X.

In den wissenschaftlichen Anwendungen, in denen das Hauptaugenmerk auf den dynamischen Eigenschaften eines Systems liegt, wird der Prozessbegriff leider oftmals synonym mit dem Systembegriff verwendet. Diese Verwechslung sollte nach Möglichkeit vermieden werden.

2.2 Prozessdarstellung

Eine erste Strukturierung und Einteilung von $\mathcal{X}^*$ als der globalen Menge aller möglichen Prozesse über dem Trägertupel (T, X) kann auf der Basis der Darstellungsmöglichkeiten der Prozesse $\Xi \in \mathcal{X}^*$ erfolgen. Denn bei der näheren Betrachtung verschiedener wissenschaftlicher Disziplinen, die sich mit dem Verhalten dynamischer Systeme beschäftigen, fällt auf, dass die Signale ξ eines Prozesses Ξ im Allgemeinen nicht explizit in Form von Zeitreihen vorliegen, sondern implizit durch modellhafte Prozessbeschreibungen $\mathbb{P}$ vorliegen:

$$\xi \in \Xi :\Leftrightarrow \bigwedge_{t \in D(\xi)} \mathbb{P}(t, \xi(t), \dots) \tag{2.8}$$

Der Ausdruck in (2.8) besagt also, dass diejenigen Signale ξ den Prozess Ξ konstituieren, die der Prozessbeschreibung $\mathbb{P}$ genügen, wobei es sich bei $\mathbb{P}$ ganz allgemein um prädikatenlogische Ausdrücke, Gleichungen (algebraische Gleichungen, Differenzialgleichungen usw.) oder Modelle (Blockschaltbilder, Automaten, usw.) handeln kann.

Aussnahmen hiervon bilden Disziplinen wie die Messtechnik oder die Signaltheorie, bei denen die Signale $\xi(t)$ oftmals in Form von gemessenen Zeitreihen vorliegen und analysiert werden.

2.2.1 Vollständigkeit

Im folgenden wird nun eine Prozessbeschreibung vorgestellt, die darauf beruht, dass man Relationen (Phasenrelationen) zwischen den Momentanwerten der Signale, den Phasen, angibt. Diese Prozessdarstellung wird Phasenstruktur genannt. Ist eine solche Prozessdarstellung mindestens auf einem Ausschnitt des Zeitbereichs T möglich, wird der Prozess als 'vollständig' bezeichnet.

Da für diese Vollständigkeitseigenschaft der Zeitbereich T von Bedeutung ist, sollen zuerst noch kurz einige Hilfsmengen eingeführt und erklärt werden, die der Strukturierung von T dienen:

- Die Menge

  $$S \subset T \tag{2.9}$$

  heißt Segment und ist eine echte Teilmenge des Zeitbereichs T.

- Das n-Tupel

  $$\underline{t} = (t_1, \dots, t_n) \in S^n \tag{2.10}$$

  mit $t_1 < t_2 < \dots < t_n$ und $\bigwedge_{t_i, t_j} (t_i \neq t_j)$ heißt Zeittupel. Ähnlich wie beim Zeitbereich T liegt beim Zeittupel $\underline{t}$ eine temporale Ordnung vor.

- Die Menge aller möglichen Zeittupel

$$S^* = \bigcup_{n \in \mathbb{N}} S^n \tag{2.11}$$

heißt Zeittupelraum. Eine Teilmenge

$$\underline{T} \subset S^* \tag{2.12}$$

heißt Zeittupelbereich.

Berücksichtigt man die Segmente $D(\xi)$ bzw. $D(\Xi)$, wobei es sich bei $D(\xi)$ gemäß (2.1) um den Definitionsbereich des Signals ξ und bei $D(\Xi) = \bigcap_{\xi \in \Xi} D(\xi)$ um den Definitionsbereich des Prozesses Ξ handelt, ergeben sich spezielle Zeittupelbereiche:

- Die Menge

$$\underline{T}_\xi = \bigcup_{n \subset \mathbb{N}} D(\xi)^n \in S^* \tag{2.13}$$

heißt Zeittupelbereich (oder Menge der Zeittupel) über dem Definitionsbereich des Signals ξ, die Menge

$$\underline{T}_\Xi = \bigcup_{n \in \mathbb{N}} D(\Xi)^n \subset S^* \tag{2.14}$$

heißt Zeittupelbereich (oder Menge der Zeittupel) über dem Definitionsbereich des Prozesses Ξ.

Wie bereits erwähnt, kann das Prozessverhaltens mit n-stelligen Relationen im Phasenraum X^n des Prozesses Ξ, den sogenannten Phasenrelationen, beschrieben werden. Diese Phasenrelationen wiederum werden durch Phasentupel gebildet. Beide Begriffe lassen sich folgendermaßen definieren:

- Das n-stellige Tupel

$$\pi_{\underline{t}}(\xi) = (\xi(t_1), \ldots, \xi(t_n)) \in X^n \tag{2.15}$$

mit $\underline{t} \in \underline{T}_\xi$, $n = |\underline{t}|$ und $\xi \in \Xi$ heißt (n-stelliges) Phasentupel oder auch $\underline{t}$-Projektion von ξ auf X.

Analog dazu kann man die $\underline{t}$-Projektion des gesamten Prozesses Ξ auf X definieren. Dann erhält man eine Menge von n-stelligen Phasentupeln und damit eine Relation $\underline{R} \subset X^n$.

- Die n-stellige Relation

$$\pi_{\underline{t}}(\Xi) = \rho(\underline{t}) = \underline{R} \subset X^n \tag{2.16}$$

mit $n = |\underline{t}|$ heißt Phasenrelation von Ξ.

Die n-stelligen Relationen $\pi_{\underline{t}}(\Xi)$ kann man für alle $\underline{t} \in \underline{T}$ zu einer Relationenfamilie zusammenfassen:

- Die Relationenfamilie

$$\pi_{\underline{T}}(\Xi) = \rho = \{\pi_{\underline{t}}(\Xi) \mid \underline{t} \in \underline{T}\} \tag{2.17}$$

heißt Phasenstruktur von Ξ.

Eine für die Charakterisierung von Prozessen wichtige Frage ist nun, ob sich ein Prozess Ξ durch eine Phasenstruktur $\pi_{\underline{T}}(\Xi)$ eindeutig darstellen lässt. Solche Prozesse sollen vollständig genannt werden. In den meisten (und auch praktisch relevanten) Fällen ist dies möglich, wobei es hier auch auf die richtige Wahl des Zeittupelbereiches $\underline{T}$ ankommt. Dies wird im folgenden anhand einiger Beispiel gezeigt.

Zunächst jedoch soll der Vollständigkeitsbegriff für Prozesse formal gefasst werden. Zu beachten ist hier, dass die Vollständigkeit vom Zeittupelbereich $\underline{T}$ abhängt, in dem die Prozessdarstellung als Phasenrelation gelten soll. Ein Prozess Ξ, der sich für alle $\underline{t} \in \underline{T}$ durch $\pi_{\underline{t}}(\Xi)$ eindeutig darstellen lässt, wird deshalb auch $\underline{T}$-vollständig genannt:

- Sei $\underline{T}$ ein Zeittupelbereich mit der Eigenschaft $\underline{T} \subset \underline{T}_\Xi$ und $\underline{T} \subset \underline{T}_{\Xi'}$ (d.h. die Zeitupelmengen $\underline{T}_\xi$ aller $\xi \in \Xi \bigcup \Xi'$ enthalten $\underline{T}$), dann heißt der Prozess Ξ genau dann $\underline{T}$-vollständig, wenn gilt:

$$\bigwedge_{\Xi'} \bigwedge_{\underline{t} \in \underline{T}} (\pi_{\underline{t}}(\Xi) = \pi_{\underline{t}}(\Xi') \Rightarrow \Xi = \Xi') \tag{2.18}$$

Aus (2.18) geht also hervor, dass es keinen weiteren Prozess Ξ' gibt, der sich von Ξ unterscheidet, aber eine gleiche Phasenstruktur aufweist. Das Prozessgesetz für einen $\underline{T}$-vollständigen Prozess lässt sich unter Verwendung der Phasenrelation $\rho(\underline{t})$ aus (2.16) allgeimein folgendermaßen formulieren:

$$\xi \in \Xi \Rightarrow \bigwedge_{\underline{t} \in \underline{T}} \big(\pi_{\underline{t}}(\xi) \in \rho(\underline{t})\big) \tag{2.19}$$

Die $\underline{T}$-Vollständigkeit eines Prozesses hängt im Allgemeinen von der Wahl von $\underline{T} \subset \underline{T}_\Xi$ ab. Nur wenn ein Prozess für kein $\underline{T}$ vollständig ist, sprechen wir von einem unvollständigen Prozess.

Eine weitere, auf dem Vollständigkeitsbegriff basierende Unterteilung von Prozessen wird möglich, wenn man nun die Zeittupel $\underline{t}$ eines Zeittupelbereichs $\underline{T} \subset S^*$ nach ihrer zeitlichen Länge einteilt. Man kann dann Prozesse, die in einem Zeittupelbereich $\underline{T}_L$ vollständige sind, dessen Zeittupel sich über eine gewisse Breite $2L$ erstrecken, L-vollständig nennen und kommt dann somit zu einem präziseren Vollständigkeitsbegriff. Dies soll im Folgenden näher erläutert werden.

Zunächst wird der Zeittupelbereich L_τ^* eingeführt, der sich dadurch auszeichnet, dass sich alle Zeittupel $\underline{t} \in L_\tau^*$ nur innerhalb einer Umgebung $L_\tau = \{t \in T \mid ||\tau - t|| \leq L\}$ um den Zeitpunkt τ befinden. Dies lässt sich formal folgendermaßen ausdrücken:

- Die Menge L_τ^* mit der Eigenschaft

$$\bigcup_{n\in\mathbb{N}} L_\tau^n \tag{2.20}$$

  ist der Zeittupelbereich mit Zeittupeln $\underline{t} = (t_1, \ldots, t_n)$, für die alle t_i mit $(i = 1, 2, \ldots, n)$ in der Umbegung L_τ liegen.

Prozesse, die nun in einem bestimmten Zeittupelbereich $\underline{T}_L \subset \bigcup_{\tau\in T} L_\tau^*$ vollständig sind, heissen dann L-vollständig.

- Ein Prozess Ξ mit der Eigenschaft

$$\xi \in \Xi \Leftrightarrow \bigwedge_{\underline{t}\in\underline{T}_L} \pi_{\underline{t}}(\xi) \in \pi_{\underline{t}}(\Xi) \tag{2.21}$$

  heißt L-vollständig.

Alle beliebigen Signale $\xi \in \Xi^*$, deren Phasentupel $\pi_{\underline{t}}(\xi)$ für alle Zeittupel $\underline{t}$ der Länge $2L$ mit der Phasenrelation $\pi_{\underline{t}}(\Xi)$ von Ξ übereinstimmen, konstituieren den L-vollständigen Prozess. Das heißt also, dass ein L-vollständiger Prozess schon durch Angabe seines Verlaufs innerhalbe eines bestimmten Zeitintervalls definiert werden kann. Dies ist von praktischer Bedeutung , denn dadurch erleichtert sich die Definition und Darstellung von Prozessen.

Mit der Definition der L-Vollständigkeit kann man nun für ein bestimmtes L eine Vollständigkeitsklasse, d.h. die Menge aller L-vollständigen Prozesse, erklären. Die folgende Liste zeigt in übersichtlicher Form einige Vollständigkeitsklassen für verschiedene L:

- Die Menge $\mathcal{V}_L = \{\Xi \mid \Xi \text{ ist } \underline{T}_L\text{-vollständig }\}$ ist die Klasse der $\underline{T}_L$-vollständigen Prozesse.
- Die Menge $\mathcal{V}_l = \{\Xi \mid \bigwedge_{L>0} (\Xi \text{ ist } \underline{T}_L\text{-vollständig })\}$ ist die Klasse der lokal-vollständigen Prozesse. Insbesondere gilt die Beziehung $\mathcal{V}_l = \bigcap_{L>0} \mathcal{V}_L$.

- Die Menge $\mathcal{V}_0 = \{\Xi \mid \Xi \text{ ist } \underline{T}_L\text{-vollständig für } L = 0\}$ ist die Klasse der im Status quo vollständigen Prozesse.

Weiterhin kann man die Menge aller vollständigen Prozesse definieren:

- Die Menge

$$\mathcal{V} = \{\Xi \mid \bigvee_{\underline{T}} (\Xi \text{ ist } \underline{T}\text{-vollständig})\} \tag{2.22}$$

  ist die Menge aller vollständigen Prozesse.

Zusammenfassend kann man folgenden Hierarchiebeziehung zwischen den bisher definierten Vollständigkeitsklassen aufstellen:

$$\mathcal{V}_0 \subset \mathcal{V}_l \subset \mathcal{V}_L \subset \mathcal{V} \subset \mathcal{X}^* \tag{2.23}$$

Das Ergebnis dieser Betrachtung scheint im Hinblick auf technische und naturwissenschaftliche Einzelwissenschaften als selbstverständlich, da ja dort alle betrachteten Prozesse mit Phasenrelationen definiert werden (oder mit Repräsentationen, die in eine äquivalente Phasenrelation umgeformt werden können) und daher vollständig sind. Für eine tiefergehende Prozesseinteilung und -charakterisierung ist dies aber durchaus notwendig und gerechtfertigt, denn die Darstellbarkeit von Prozessen mit einer Phasenrelation darf eigentlich nicht als selbstverständlich angenommen werden.

2.2.2 Beispiele

Um die Ergebnisse des vorigen Abschnitts 2.2.1 näher zu verdeutlichen, sollen nun einige Beispiele zu dieser Thematik aufgeführt und diskutiert werden:

1. Abbildung 2.1 zeigt den Verlauf eines Signales $\xi \in \Xi$. Die Trägermengen sind $X = \{a, b\}$ und $T = \mathbb{N}$.

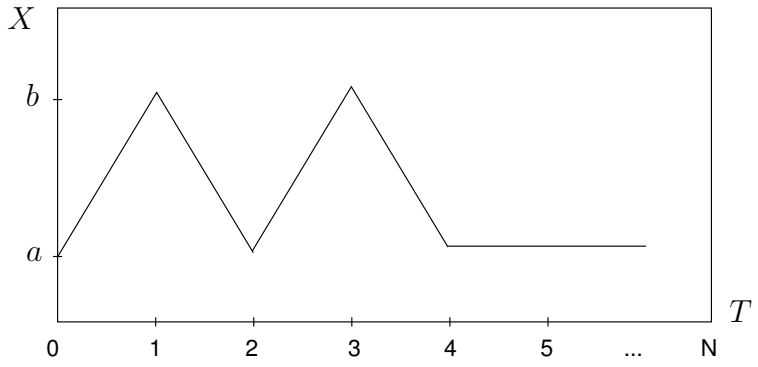


Abbildung 2.1: Signalverlauf von $\xi \in \Xi$

Der Prozess Ξ ist folgendermaßen bestimmt:

$$\xi \in \Xi \quad \Leftrightarrow \quad \xi = \begin{cases} b & t = 1, 3 \\ a & \text{sonst} \end{cases} \tag{2.24}$$

mit $D(\xi) = D(\Xi) = \{0, 1 \ldots, N\}$.

Für verschiedene Zeittupelbereiche $\underline{T}$ soll nun die Vollständigkeit untersucht werden:

a) $\underline{T} = \{(0, 1, 2), (2, 3, 4)\}$, somit gilt $\underline{T} \subset \underline{T}_\xi = \underline{T}_\Xi$. Die Phasentupel $\pi_{\underline{t}}(\xi)$ sind unabhängig von $\underline{t}$ und lauten:

$$\pi_{\underline{t}}(\xi) = (a, b, a) \quad \text{für alle } \underline{t} \in \underline{T}$$

Da der Prozess Ξ in diesem einfachen Beispiel nur aus einem Signal ξ besteht und das Phasentupel unabhängig von der Zeit ist, gilt für die Phasenstruktur:

$$\rho = \pi_{\underline{T}}(\Xi) = \{(a, b, a)\} \tag{2.25}$$

Wie im folgenden gezeigt wird, ist Ξ nicht $\underline{T}$-vollständig.
Sei ein Prozess Ξ' folgendermaßen definiert:

$$\xi' \in \Xi' \quad \Leftrightarrow \quad \xi' = \begin{cases} a & t = 0, 2, 4 \\ b & \text{sonst} \end{cases} \tag{2.26}$$

mit $D(\xi) = D(\Xi) = \{0, 1 \ldots, N\}$. Auch hier gilt:

$$\underline{T} \subset \underline{T}_{\Xi'}$$

$$\pi_{\underline{t}}(\xi') = (a, b, a) \quad \text{für alle } \underline{t} \in \underline{T}$$

$$\rho' = \pi_{\underline{T}}(\Xi') = \{(a, b, a)\}$$

somit gilt:

$$\bigwedge_{\underline{t} \in \underline{T}} \pi_{\underline{t}}(\Xi) = \pi_{\underline{t}}(\Xi') \text{ und } \Xi \neq \Xi'$$

Dies widerspricht der Vollständigkeitsdefinition in (2.18). Somit ist Ξ (und auch Ξ') $\underline{T}$-unvollständig.
Es lässt sich nun leicht ein Zeittupelbereich finden, für den Ξ vollständig wird (und damit durch ρ eindeutig spezifizierbar).

b) Mit $\underline{T} = \{\underline{t}_1 = (0,1,2), \underline{t}_2 = (2,3,4), \underline{t}_3 = (5,\ldots,N)\}$ lauten die Phasentupel:

$$\pi_{\underline{t}}(\xi) = \begin{cases} (a,b,a) & \text{für} \quad \underline{t} = \underline{t}_1, \underline{t}_2 \\ (a,\ldots,a) & \text{für} \quad \underline{t} = \underline{t}_3 \end{cases}$$

Für diesen Zeittupelbereich $\underline{T}$ unterscheidet sich nun die Phasenstruktur ρ_Ξ von $\rho_{\Xi'}$ und auch von allen anderen möglichen Phasenstrukturen. Der Prozess Ξ ist damit vollständig.

2. Das nächste Beispiel zeigt einen L-vollständigen Prozess $\Xi \in \mathcal{V}_L$, der folgendermaßen definiert ist:

$$T = \mathbb{Z}, \quad X = \mathbb{R} \quad \text{(Trägermengen)}$$

$$L \in \mathbb{N}, \quad \underline{T}_L = \bigcup_{t \in S} (t-L, \ldots, t+L) \quad \text{(Zeittupelbereich)}$$

$$\xi \in \Xi \quad \Leftrightarrow \quad \bigwedge_{t \in S} (\xi(t-L), \xi(t-L+1), \ldots, \xi(t+L)) \in \pi_{\underline{T}}(\Xi)$$

Das Zeitsegment S ist ein Intervall aus $\mathbb{Z}$ und ist natürlich so zu wählen, dass $\underline{T}_L(S) \subset \underline{T}_\Xi$.

Der Prozess wird durch die Phasenrelation $\pi_{\underline{T}}(\Xi)$ eindeutig bestimmt. Und da alle Zeittupel $\underline{t} \in \underline{T}_L$ die 'Breite' $2L$ haben, ist Ξ ein L-vollständiger Prozess.

Zur besseren Verdeutlichung des Prozesses kann man eine lokale Verhaltensfunktion ψ einführen, die der Definition von $\pi_{\underline{T}}(\Xi)$ dient:

$$(\xi(t), \xi(t+1), \ldots, \xi(t+2L)) \in \pi_{\underline{T}}(\Xi)$$

$$\Leftrightarrow$$

$$\psi\big((t, \ldots, t+2L), (\xi(t), \ldots, \xi(t+2L))\big) = 0$$

Bei diesem Beispiel handelt sich also um einen Differenzenprozess, der durch eine homogene Differenzengleichung ψ der Ordnung $n = 2L$ dargestellt werden kann.

3. Gegeben sei ein reeller, kontinuierlicher Prozess Ξ mit

$$T = \mathbb{R}, \quad X = \mathbb{R} \quad \text{(Trägermengen)}$$

$$\xi \in \Xi \quad \Leftrightarrow \quad K\dot{\xi} + \xi = 1 \quad \text{und} \quad \xi(t \le 0) = 0$$

Die Lösung der Differenzialgleichung in ξ lautet $\xi = (1 - e^{-\frac{t}{K}})$. Somit gilt also:

$$\bigwedge_{L>0}\Big(\xi \in \Xi \Leftrightarrow \bigwedge_{\underline{t}\in\underline{T}_L} \pi_{\underline{t}}(\xi) = \big((1 - e^{-\frac{t_1}{K}}), \ldots, (1 - e^{-\frac{t_n}{K}})\big)\Big)$$

$$\text{mit} \quad \underline{t} = (t_1, \ldots, t_n)$$

Der Prozess Ξ ist somit lokal vollständig ($\Xi \in \mathcal{V}_l$). Solche reellen, kontinuierlichen Prozesse Ξ, die lokal vollständig sind und deren Signale ξ differenzierbar sind, nennt man auch Differenzialprozesse. Solche Prozessse sind immer die Lösungsmenge einer lokalen Verhaltensgleichung der Form

$$\psi_0\big(t, \xi(t), \dot{\xi}(t), \ldots, \xi^{(n)}\big) = 0$$

Man sieht insbesondere, dass ψ_0 nur von t und nicht von $\underline{t}$ abhängt. Lokale Verhaltensgleichungen, in denen auch $\underline{t}$ vorkommt, sind z.B. Differenzialgleichungen mit Laufzeit. Anhand der Beispiele 2) und 3) zeigt sich, dass die in den unterschiedlichen wissenschaftlichen Dispziplinen durch Differenzen- oder Differnzialgleichungen dargestellten Prozesse der Klasse der lokal-vollständigen ($\mathcal{V}_l$) oder der L-vollständigen ($\mathcal{V}_L$) Prozesse angehören.

2.3 Temporale Wechselwirkung

Im vorangegangenen Kapitel wurde das Problem der Prozessdarstellung (Vollständigkeit) untersucht und als Klassifikationskriterium für Prozesse genutzt. Anhand des Vollständigkeitskriteriums kann gezeigt werden, wie ein Prozess Ξ durch seine Phasenstruktur ρ zweifelsfrei und eindeutig dargestellt werden kann, bzw. welches Mindestmaß an Informationen dafür notwendig ist. Im folgenden nun werden die unterschiedlichen Aspekte der temporalen Wechselwirkung (Verhalten) von Prozessen erläutert. Auch diese Aspekte können zu einer grundlegenden Prozessklassifikation herangezogen werden. Dazu werden bereits hinlänglich bekannte, aber oft unterschiedlich verstandene Begriffe wie 'reversibel' und 'determiniert' definiert. Weiterhin werden die Beziehungen zwischen Vollständigkeits- und Verhaltenseigenschaften von Prozessen erläutert.

2.3.1 Wechselwirkungsrelation

Zu Beginn werden erst einige Begriffe eingeführt und definiert. Zunächst folgt eine formale Definition der Begriffe (Prozess-)Vergangenheit und (Prozess-)Zukunft. Daran anschließend wird gezeigt, wie sich die temporale Wechselwirkung eines Prozesses durch eine Wechselwirkungsrelation beschreiben lässt. Da diese Wechselwirkungsrelation zeitabhängig ist, wird eine Kopplungsfunktion definiert.

- Die Menge $T^\tau \subset T$ mit

$$T^\tau = \{t \mid t \in T \wedge t < \tau\} \tag{2.27}$$

heißt Vergangenheit von τ. Ist in diese Vergangenheit τ selbst mit eingeschlossen, ergibt dies die Menge $\overline{T}^\tau$ mit $\overline{T}^\tau = \{t \mid t \in T \wedge t \leq \tau\}$.

- Die Menge $T_\tau \subset T$ mit

$$T_\tau = \{t \mid t \in T \wedge t > \tau\} \tag{2.28}$$

heißt Zukunft von τ. Ist in diese Zukunft τ selbst mit eingeschlossen, ergibt dies die Menge $\overline{T}_\tau$ mit $\overline{T}_\tau = \{t \mid t \in T \wedge t \geq \tau\}$.

Schränkt man nun den Definitionsbereich von Signalen oder Prozessen auf solche Vergangenheits- oder Zukunftsmengen ein, so spricht man auch von Signal- oder Prozessvergangenheit bzw. Signal- oder Prozesszukunft.

- Das auf T^τ oder T_τ eingeschränkte Signal $\xi|T^\tau$ oder $\xi|T_\tau$ nennt man Signalvergangenheit oder Signalzukunft.
- Analog heißt der auf T^τ oder T_τ eingeschränkte Prozess $\Xi|T^\tau = \bigcup_{\xi \in \Xi} \xi|T^\tau$ Prozessvergangenheit oder Prozesszukunft ($\Xi|T_\tau = \bigcup_{\xi \in \Xi} \xi|T_\tau$).

Diese Definitionen ermöglichen also die Einteilung der Trägermenge T in eine Zukunft und eine Vergangenheit bezüglich τ. Dies erweist sich als sinnvoll, denn sowohl die Anschauung also auch die Erfahrung aus unterschiedlichen Einzelwissenschaften führen zu der Annahme, dass sich der Verlauf eines Prozesses in der Vergangenheit im Allgemeinen auch auf dessen zukünftigen Verlauf auswirkt, dass also eine temporale Wechselwirkung existiert. Wie diese temporale Wechselwirkung formal gefasst und qualifiziert werden kann, wird nun gezeigt.

Eine temporale Wecheselwirkung zwischen zwei Signalen ξ und ξ' eines Prozesses Ξ äußert sich darin, dass die Zukunft des einen Signals mit der Vergangenheit des anderen verträglich ist, dass also ein Signal $\xi'' \in \Xi$ existiert mit $\xi''|T^\tau = \xi|T^\tau$ und $\xi''|T_\tau = \xi'|T_\tau$ bzw. unter Verwendung des Konkatenationsproduktes $\xi \overset{\tau}{\circ} \xi' = \xi'' \in \Xi$, wobei das Konkatenationsprodukt folgendermaßen definiert ist:

$$\xi \overset{\tau}{\circ} \xi' = \begin{cases} \xi & \text{für } t \leq \tau \\ \xi' & \text{für } t > \tau \end{cases} \tag{2.29}$$

Dieser Sachverhalt kann mit der Wechselwirkungsrelation, die folgendermaßen definiert ist, beschrieben werden.

- Die binäre Relation $W_\tau \subset \Xi \times \Xi$ mit

$$W_\tau = \left\{(\xi, \xi') \in \Xi \times \Xi \mid \xi \overset{\tau}{\circ} \xi' \in \Xi\right\} \tag{2.30}$$

heißt (temporale) Wechselwirkungsrelation. Der Zeitpunkt τ muss hierbei natürlich in $D(\xi)$ und in $D(\xi')$ liegen, also $\tau \in D(\xi) \cap D(\xi')$. Wenn für $(\xi, \xi') \in W_\tau$ gilt, nennt man ξ und ξ' auch τ-verträglich.

W_τ ist im Allgemeinen reflexiv ($(\xi, \xi) \in W_\tau$) und transitiv ($(\xi, \xi') \in W_\tau \wedge (\xi, \xi'') \in W_\tau \Rightarrow (\xi, \xi'') \in W_\tau$), aber nicht symmetrisch ($(\xi, \xi') \in W_\tau \not\Rightarrow (\xi', \xi) \in W_\tau$).

Wie aus der Definition ersichtlich ist, ist die Wechselwirkungsrelation W_τ also abhängig vom Zeitpunkt τ. Diese zeitliche Abhängigkeit kann man nun als Abbildung von der Menge aller zulässigen Zeitpunkte τ auf die Menge aller Relation W_τ ausdrücken. Diese Abbildung heißt dann Kopplungsfunktion und ist folgendermaßen definiert:

- Die Abbildung

$$\kappa : \overline{D}(\Xi) \mapsto \mathsf{P}(\Xi \times \Xi) \tag{2.31}$$

mit $\kappa(\tau) = W_\tau \subset \Xi \times \Xi$ heißt (temporale) Kopplungsfunktion.

Da ja W_τ reflexiv ist und jedes Signal ξ zumindest mit sich selbst verträglich ist, ist κ auf $\overline{D}(\Xi) = \bigcup_{\xi \in \Xi} D(\xi)$ definiert. Der Wertebereich von κ hingegen ist die Menge aller Teilmengen von $\Xi \times \Xi$.

Die temporale Kopplungsfunktion κ kann also als ein qualitatives Maß für die momentane Wecheslwirkung innerhalb eines Prozesses angesehen werden. Dabei kann diese Wechselwirkung für jeden Zeitpunkt τ zwischen zwei Extremen schwanken:

$$I_\Xi \subseteq \kappa(\tau) \subseteq A_\Xi \tag{2.32}$$

Die Menge $I_\Xi = \left\{(\xi, \xi) \mid \xi \in \Xi\right\}$ ist die Identitätsrelation, die Menge $A_\Xi = \Xi^2$ ist die Allrelation von Ξ. Dies bedeutet also, dass die Wechselwirkung entweder auf identische Signale beschränkt und das Verhalten damit streng funktional ist, oder jedes Signal mit jedem und zu jeder Zeit in Wechselwirkung treten kann und das Verhalten damit chaotisch ist. Dies führt zu den folgenden Definitionen:

- Ein Prozess $\Xi \in \mathcal{X}^*$ heißt chaotischer Prozess, wenn für die temporale Kopplungsfunktion κ gilt:

$$\bigwedge_{\tau \in \overline{D}(\Xi)} \kappa(\tau) = A_\Xi \tag{2.33}$$

- Ein Prozess $\Xi \in \mathcal{X}^*$ heißt bifunktionaler Prozess, wenn für die temporale Kopplungsfunktion κ gilt:

$$\bigwedge_{\tau \in \overline{D}(\Xi)} \kappa(\tau) = I_\Xi \tag{2.34}$$

Problematisch wird diese Definition jedoch in den Fällen, in denen $I_\Xi = A_\Xi$ ist, also z.B. für einen Prozess $\Xi = \{\xi_0\}$, der nur aus einem einzigen Signal besteht. Dieser Prozess kann natürlich formal sowohl chaotisch als auch bifunktional bezeichnet werden. Jedoch eignet sich rein intuitiv der Begriff 'bifunktional' in diesem Fall besser.

2.3.2 Gedächtnisprozesse

Im folgenden wird untersucht, wie man vom dynamischen Verhalten eines Prozesses innerhalb eines bestimmten Zeitbereichs, des sogenannten Gedächtnisbereichs, auf dessen Wechselwirkung schließen kann. Um diesen Sachverhalt formal zu fassen, ist die bereits auf Seite 12 definiert Umgebung $L_\tau = \{t \in T \mid ||\tau - t|| \leq L\}$ notwendig, die hier im Sinnes eines Gedächtnisbereiches verwendet wird.

Das Gedächtnis eines Prozesses äußert sich nun darin, dass zwei Signale ξ und ξ', die über einen ganzen Gedächtnisbereich L_τ identisch sind, in der Prozesszukunft 'miteinander kompatibel' und damit τ-verträglich sind. Prozesse, deren Signale diese Eigenschaft besitzen, nennt man dann Gedächtnisprozess mit dem Gedächtnisbereich oder auch Gedächtnis L_τ.

- Ein Prozess Ξ mit der Eigenschaft

$$\xi|\overline{L}_\tau = \xi'|\overline{L}_\tau \Rightarrow \xi \overset{\tau}{\circ} \xi' \in \Xi \tag{2.35}$$

für alle $\xi, \xi' \in \Xi$ und $\tau \in D(\kappa) = \overline{D}(\Xi)$ heißt Gedächtnisprozess. Der Bereich $\overline{L}_\tau = L_\tau \cap D(\xi) \cap D(\xi')$ bedeutet hier die Beschränkung des Gedächtnisbereichs L_τ auf die Definitionsbereiche der Signale ξ und ξ'.

Wichtig in(2.35) ist die Implikation von $\xi|\overline{L}_\tau = \xi'|\overline{L}_\tau$ auf $\xi \overset{\tau}{\circ} \xi' \in \Xi$, da zwei Signale ξ und ξ' durchaus τ-verträglich sein können, ohne dass sie im Gedächtnisbereich gleich verlaufen. Dies findet auch in der Definition der Kopplungsfunktion κ_L seinen Ausdruck:

- Die temporale Wechselwirkung eines Gedächtnisprozesses wird durch die Kopplungsfunktion κ_L mit

$$(\xi|\overline{L}_\tau = \xi'|\overline{L}_\tau) \Leftrightarrow (\xi, \xi') \in \kappa_L \Rightarrow \xi \overset{\tau}{\circ} \xi' \in \Xi \tag{2.36}$$

für alle $\xi, \xi' \in \Xi$ und $\tau \in D(\kappa) = \overline{D}(\Xi)$ qualitativ beschrieben.

Mit der Kopplungsfunktion κ_L ist auch die Definition der Klasse von Gedächtnisprozessen möglich. Diese Prozesseklasse soll im weiteren mit $\mathcal{X}_L$ bezeichnet werden und ist die Menge aller Prozesse Ξ, für die eine Kopplungsfunktion κ_L gilt (deren Signale τ-verträglich sind, wenn diese Signale in einem Gedächtnisbereich der Länge $2L$ gleich verlaufen).

- Die Menge $\mathcal{X}_L$ der Prozesse $\Xi_L \in \mathcal{X}_L$, für die gilt:

$$(\xi, \xi') \in \kappa_L \Rightarrow \xi \overset{\tau}{\circ} \xi' \in \Xi_L \tag{2.37}$$

  heißt Klasse der Gedächtnisprozesse mit 'Gedächtnisdauer' $2L$. Da sich die Klasse durch ein bestimmtes Prozessverhalten auszeichnet, handelt es sich hier um eine Verhaltensklasse.

An dieser Stelle sei bemerkt, dass κ_L im Gegensatz zur allgemeinen Kopplungsfunktion κ eine symmetrische Relation ist $((\xi, \xi') \in \kappa_L \Rightarrow (\xi', \xi) \in \kappa_L)$ und somit eine Aquivalenzrelation auf Ξ_L ist.

Neben der Einteilung von Prozessen in Vollständigkeitsklassen (2.23) ist also auch eine Einteilung in Verhaltensklassen möglich. Und analog zu den Vollständigkeitsklassen kann man auch die Verhaltensklassen nach der Gedächtnisdauer einteilen.

- Die Menge $\mathcal{X}_0$ ist die Klasse der Prozesse ohne Gedächtnis. Das Verhalten solcher Prozesse hängt also nur vom aktuellen Zustand zur Zeit τ ab. Die Zustände aus der früheren Vergangenheit haben keinen Einfluß auf die Zukunft. Hier handelt es sich also um Markov-Prozesse. Ihr Verhalten lässt sich im diskreten mit Automaten und im kontinuierlichen mit Zustandsdifferenzialgleichungen darstellen. Mit $L = 0$ wird aus (2.31) folgende Gleichung:

$$(\xi(\tau) = \xi'(\tau)) \Leftrightarrow (\xi, \xi') \in \kappa_0 \tag{2.38}$$

- Die Menge $\mathcal{X}_l$ ist die Klasse der Prozesse mit lokalem Gedächtnis, d.h.

$$\mathcal{X}_l = \bigcap_{L>0} \mathcal{X}_L \tag{2.39}$$

  Hierzu zählen insbesondere die Differenzialprozesse aus Abschnitt 2.2.2.

- Die Menge $\mathcal{X}_L$ ist die Klasse der Prozesse mit konstantem Gedächtnis der Länge $2L$, siehe (2.37).

Ähnlich wie in (2.23) existiert zwischen den einzelnen Klassen eine Hierarchiebeziehung:

$$\mathcal{X}_0 \subset \mathcal{X}_l \subset \mathcal{X}_L \tag{2.40}$$

Die Vollständigkeits- und Verhaltensklassen kann man nun in Beziehung zueinander setzen, indem man von bestimmten Vollständigkeitseigenschaften auf Verhaltenseigenschaften schließt. Man kann zeigen, dass folgende Implikationen gelten:

$$\Xi \in \mathcal{V}_0 \Rightarrow \Xi \in \mathcal{X}_0 \qquad \Xi \in \mathcal{V}_l \Rightarrow \Xi \in \mathcal{X}_l \qquad \Xi \in \mathcal{V}_L \Rightarrow \Xi \in \mathcal{X}_L \tag{2.41}$$

2.3.3 Determinierte und Reversible Prozesse

Diese vorgestellten Prozessklassen ermöglichen eine grundlegende Einteilung aller im automatisierungstechnischen Umfeld relevanten Prozesse. Darüber hinaus werden jetzt noch die Begriffe 'determiniert' und 'reversibel' eingeführt, die in unterschiedlichen Wissenschaftsdisziplinen verbreitet sind und auch für automatisierungstechnische Systeme nützliche Klassifizierungen ermöglichen. Dabei sollen diese Begriffe hier systemtheoretisch auf der Basis der bereits erläuterten temporalen Wechselwirkung definiert werden.

Wenn man einen Prozess als 'determiniert' bezeichnet, wird damit gemeinhin ausgedrückt, dass eine starke temporale Wechselwirkung zwischen Vergangenheit und Zukunft vorliegt: Der zukünftige Verlauf eines Prozesses kann aus seiner Vergangenheit vollständig vorhergesagt werden (wobei zunächst noch unklar sei, über welchen Zeitraum der Vergangheitsverlauf bekannt sein muss). Dies führt dann auf folgende Definition:

- Ein Prozess Ξ mit der Eigenschaft

$$(\xi \overset{\tau}{\circ} \xi') \in \Xi \Rightarrow (\xi|T_\tau = \xi'|T_\tau) \tag{2.42}$$

  für alle $\xi, \xi' \in \Xi$ und $\tau \in D(\xi) \cap D(\xi')$ (nur für diese τ ist das Konkatenationsprodukt definiert) heißt determiniert.

Solche Prozesse bezeichnet man auch als 'zukunftsfunktional'.

Mit einer analogen Überlegung kommt man zur Definition 'revesibler' Prozesse:

- Ein Prozess Ξ mit

$$(\xi \overset{\tau}{\circ} \xi') \in \Xi \Rightarrow (\xi|T^\tau = \xi'|T^\tau) \tag{2.43}$$

  für alle $\xi, \xi' \in \Xi$ und $\tau \in D(\xi) \cap D(\xi')$ heißt reversibel.

Solche Prozesse bezeichnet man auch als 'vergangenheitsfunktional'.

Determinierte und reversible Prozesse kann man ganz allgemein 'funktional' nennen. Sie zeichnen sich durch eine starke temporale Wecheselwirkung aus, die es ermöglicht, aus der Vergangenheit eindeutig auf die Zukunft zu schließen (oder umgekehrt). Bei determinierten Prozessen können Signale mit gleicher Vergangenheit nicht eine verschiedene Zukunft haben; bei reversiblen Prozesse können Signale mit gleicher Zukunft nicht unterschiedliche Vergangenheit haben. Prozesse, die sowohl reversibel als auch determiniert sind, heißen bifunktional, siehe auch (2.34).

Die Begriffe 'determiniert' und 'reversibel' sind zu den Verhaltensklassen prinzipiell orthogonal. Beide Prozesseigenschaften können deshalb miteinander kombiniert werden. Damit ergibt sich eine Möglichkeit, Prozesse aussagekräftig zu charakterisieren.

- Ein Prozess $\Xi \in \mathcal{X}^*$ mit der Eigenschaft

$$(\xi, \xi') \in \kappa(\tau) \Rightarrow \xi|T_\tau = \xi'|T_\tau \tag{2.44}$$

für alle $\xi, \xi' \in \Xi$ und $\tau \in D(\kappa) = \overline{D}(\Xi)$ heißt determinierter Prozess der Verhaltensklasse $\mathcal{X}_\kappa$, in Zeichen

$$\Xi \in \mathcal{X}_\kappa^> \tag{2.45}$$

- Ein Prozess $\Xi \in \mathcal{X}^*$ mit der Eigenschaft

$$(\xi, \xi') \in \kappa(\tau) \Rightarrow \xi|T^\tau = \xi'|T^\tau \tag{2.46}$$

für alle $\xi, \xi' \in \Xi$ und $\tau \in D(\kappa) = \overline{D}(\Xi)$ heißt reversibler Prozess der Verhaltensklasse $\mathcal{X}_\kappa$, in Zeichen

$$\Xi \in \mathcal{X}_\kappa^< \tag{2.47}$$

2.3.4 Beispiele

Es folgen erneut einige Beispiele, die die vorgestellten Zusammenhänge verdeutlichen sollen.

1. Der Verlauf des Aktienskurses in Deutschland werde beschrieben durch ein einfaches Börsengesetz: 'Der DAX bleibt nicht länger als einen Tag unter 4000 Punkte'. Die folgende Abbildung 2.2 zeigt den möglichen Verlauf zweier Signale ξ und ξ'.

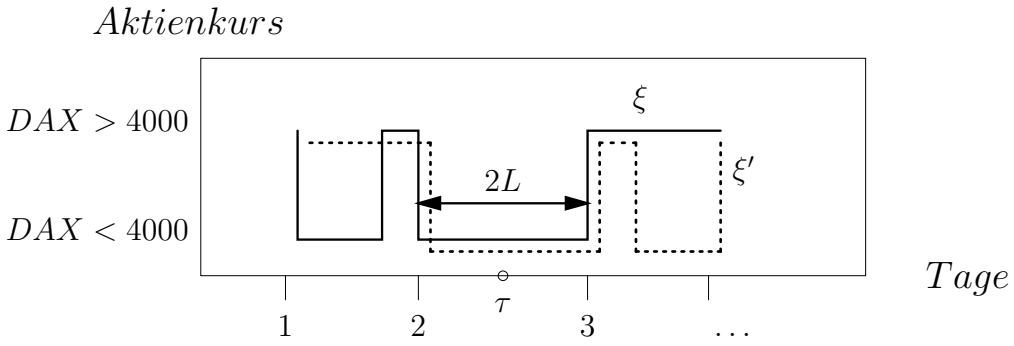


Abbildung 2.2: Gedächtnisprozess $\Xi \in \mathcal{X}_L$

Die Trägermengen von Ξ seien $X = \{< 4000, > 4000\}$ und $T = \mathbb{R}$. Der Prozess gehorcht folgendem Prozessgesetz und ist damit L-vollständig:

$$\xi \in \Xi \Leftrightarrow \xi \in \Xi^* \wedge \left(\pi_{\underline{t}}(\xi) = \{< 4000\}^{|\underline{t}|} \Rightarrow \underline{t} \in \underline{T}_L \text{ mit } 2L = 1Tag\right) \tag{2.48}$$

Die Menge Ξ^* ist nach (2.4) der Signalraum über T, X. Nach (2.41) ist $\Xi \in \mathcal{X}_L$, also ein Prozess mit konstantem Gedächtnis $2L$. Dann muss nach (2.31) und (2.37) auch gelten:

$$(\xi|\overline{L}_\tau = \xi'|\overline{L}_\tau) \Leftrightarrow (\xi, \xi') \in \kappa_L \Rightarrow \xi \overset{\tau}{\circ} \xi' \in \Xi \tag{2.49}$$

Zwei Signale mit gleichem Verlauf in einem Gedächtnisbereich $2L$ sind also τ-verträglich. Dass dies gilt, geht anschaulich aus der folgenden Abbildung 2.3:

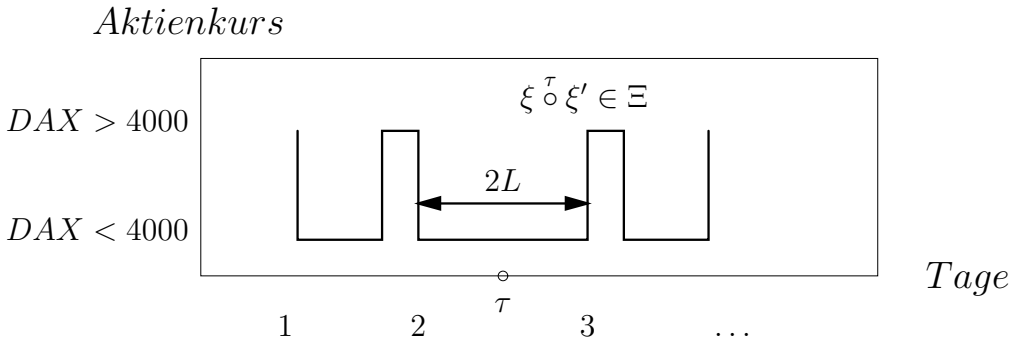


Abbildung 2.3: τ-verträgliche Signale

Der Prozess ist nicht-determiniert und irreversibel, da zwei Signale, die im Gedächtnisbereich gleich Verlaufen, nicht automatisch auch gleiche Zukunft oder Vergangenheit haben.

2. Zur Veranschaulichung der Reversibilität wird nun ein Markov-Prozess Ξ_0 betrachtet, der durch folgenden autonomen, endlichen Zustandsautomaten mit der Anfangsbedingung

$$\bigwedge_{\xi \in \Xi_0} \xi(0) = a$$

definiert ist, siehe Abbildung 2.4.

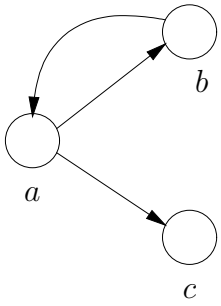


Abbildung 2.4: reversibler Automat

Der Automat wird vollständig durch eine Phasenrelation der Form

$$\pi_{\underline{t}}(\Xi) = \{(a,b),(a,c),(b,a)\} \qquad \bigwedge_{\tau \in D(\Xi)=\mathbb{Z}} \underline{t} = (\tau, \tau+1) \tag{2.50}$$

beschrieben (Vollständigkeit).

Da der Prozess ein Gedächtnis der Länge $L = 0$ hat, handelt es sich nach (2.38) um einen Markov-Prozess:

$$(\xi(\tau) = \xi'(\tau)) \Rightarrow \xi \overset{\tau}{\circ} \xi' \in \Xi_0 \tag{2.51}$$

Alle Signale beginnen mit einer (a,b)-Folge beliebiger Länge. Alle Signalpaare (ξ, ξ') haben deshalb bis zu dem Zeitpunkt τ, an dem ein Signal in c terminiert, die gleiche Vergangenheit $\xi \mid T^\tau = \xi' \mid T^\tau$. Der Prozess ist reversibel. Da der Zeitpunkt τ beliebig ist, ist der Prozess nicht determiniert. Hier handelt es sich also um einen reversiblen, nicht-determnierten Markov-Prozess:

$$\Xi_0 \in \mathcal{X}_0^< \tag{2.52}$$

(2.52) gilt unabhängig von der Anfangsbedingung. Die Reversibilitäts- und Determiniertheitseigenschaften von Zustandsautomaten lassen leicht graphentheoretisch ableiten:

- sobald ein Zustand mehr als eine Eingangskante hat, ist der Automat irreversibel
- sobald ein Zustand mehr als eine Ausgangskante hat, ist der Automat nicht-determiniert

Dies lässt sich auch leicht in Abbildung 2.4 nachvollziehen.

3. Gegeben sei der folgende Differenzialprozess Ξ mit den Trägermengen $X = \mathbb{R}$ und $T = \mathbb{R}^+$, der durch folgende Differenzialgleichung beschrieben wird:

$$\xi \in \Xi \quad \Leftrightarrow \quad \ddot{\xi} = (1-y^2)\dot{y} - y \quad \text{und} \quad \xi(t=0) \in [a,b] \quad \dot{\xi}(t=0) = 0 \tag{2.53}$$

Der Verlauf von ξ für verschiedene Anfangsbedingungen ist in der nächsten Abbildung 2.5 gezeigt. Ξ ist nach Abschnitt 2.2.2 ein lokal-vollständiger Prozess und damit nach (2.41) auch ein Prozess mit lokalem Gedächtnis, also $\Xi \in \mathcal{X}_l$. Weiterhin ist der Prozess nach (2.34) bifunktional, also reversibel und determiniert, denn alle Signale sind nur mit sich selbst τ-verträglich, gerade auch an den Stellen $\tau_{i,j}$ mit $\xi_i(\tau_{i,j}) = \xi_j(\tau_{i,j})$, an denen sich die Signale paarweise schneiden. Denn alle Signale der Form

$$\xi_m = \xi_i \overset{\tau_{i,j}}{\circ} \xi_j \quad \text{und} \quad \xi_i \neq \xi_j \tag{2.54}$$

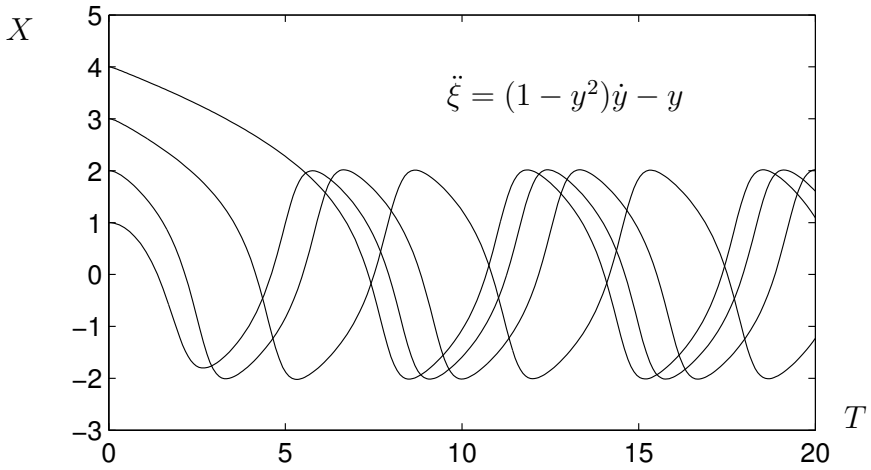


Abbildung 2.5: Differenzialprozess

sind an den Stellen $\tau_{i,j}$ nicht differenzierbar und können damit keine Lösung von (2.53) sein. Der Prozess Ξ hat damit ein lokales Gedächtnis und ist bifunktional.

$$\Xi \in \mathcal{X}_l^{<>} \tag{2.55}$$

4. Wie im letzten Beispiel gesehen haben Differenzialprozesse die Eigenschaft $\Xi \in \mathcal{X}_l$. Differenzialprozesse können aber auch Markov-Prozesse sein, wie anhand des folgenden Beispiels gezeigt wird.

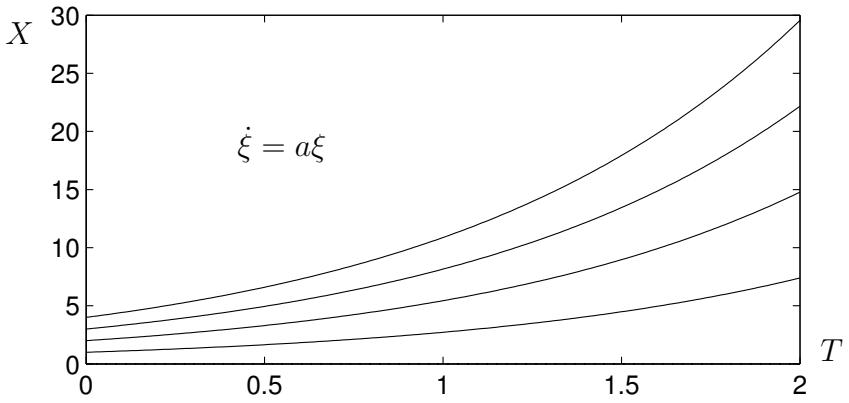


Abbildung 2.6: Exponenzialprozess

Die Signale $\xi \in \Xi$ sind durch die Lösung der Differenzialgleichung $\dot{\xi} = a\xi$ gegeben

$$\xi \in \Xi \quad \Leftrightarrow \quad \xi(t) = c \cdot e^{at} \quad \text{und} \quad \xi(t = 0) \in X_0 \subset \mathbb{R} \tag{2.56}$$

wobei die Anfangsbedingungen X_0 berücksichtigt werden müssen.

Hier handelt es zunächst um einen Differenzialprozess mit $\Xi \in \mathcal{X}_l$, siehe vorheriges Beispiel. Da sich die Prozessrealisierungen ξ aber nie schneiden, gilt:

$$\xi(\tau) = \xi'(\tau) \quad \Rightarrow \quad \xi = \xi' \quad \Rightarrow \quad \xi \overset{\tau}{\circ} \xi' \in \Xi \tag{2.57}$$

Nach (2.38) handelt es sich also auch um einen Markov-Prozess. Dies bestätigt die Inklusion aus (2.40). Sobald aber zwei Signale $\xi, \xi' \in \Xi$ τ-verträglich sind, sind sie auch gleich:

$$\xi = \xi' \Leftarrow \xi \overset{\tau}{\circ} \xi' \in \Xi \tag{2.58}$$

Dies wiederum bedeutet nach (2.34), dass Ξ bifunktional ist. Zusammenfassend lässt sich der Prozess nach (2.56) als bifunktionaler Markov-Prozess charakterisieren, $\Xi \in \mathcal{X}_0^{<>}$. Bei solchen Prozessen kann man von einem gegebenen $\xi(t)$ eindeutig auf den zukünftigen und auch vergangenen Signalverlauf ξ geschlossen werden. Zusammenfassend lassen sich die Erkenntnisse über bifunktionale Markovprozesse folgendermaßen formulieren:

$$\Xi \in \mathcal{X}_0^{<>} \Leftrightarrow \big(\xi(\tau) = \xi'(\tau) \Leftrightarrow \xi = \xi'\big) \tag{2.59}$$

für alle $\xi, \xi' \in \Xi$ und $\tau \in D(\xi) \cap D(\xi')$.

Die Signale bifunktionaler Markov-Prozesse schneiden sich also nie. Tatsächlich gibt es in Naturwissenschaft und Technik eine Reihe von Beispielen für bifunktionale Markov-Prozesse. Grundsätzlich lassen sich die dort betrachteten Prozesse als vollständige, irreversible, nicht-determinierte Prozess mit Gedächtnis ($\mathcal{X}_L$) charakterisieren.

2.4 Kooperative Wechselwirkung

Aus der Erfahrung heraus ist in unterschielichen wissenschaftlichen Disziplinen bekannt, dass nicht nur eine temporale Wechselwirkung innerhalb eines Prozessen stattfinden kann, sondern auch eine kooperative Wechselwirkung zwischen einem oder mehreren Prozessen. Man spricht dann auch von 'Prozesskooperation', die an der Kooperation teilnehmenden Prozesse nennt man 'kooperierende Prozesse'.

Diesem Phänomen muss natürlich auch von Seiten einer formalen Prozesstheorie Rechnung getragen werden. Deshalb soll im nun folgenden Kapitel gezeigt werden, wie in der Prozesstheorie nach [Wunsch 2000] die kooperative Wechselwirkung formal gefasst werden kann. Diese kooperative Wechselwirkung wird zunächst zwischen zwei Prozessen untersucht, um dann auf eine beliebige Anzahl kooperierender Prozesse übertragen zu werden. Weiterhin werden in diesem Zusammenhang unterschiedlichen Bedeutungen des Begriffs 'Kausalität' erläutert und miteinander verglichen. Abschließend werden einige für die Simulations- und Analysemethodik bedeutende Merkmale der kausalen Kopplungsrelationen, mit der Prozesskooperation beschrieben werden kann, untersucht (mathematische Struktur, Berechenbarkeit).

2.4.1 Prozesskooperation

Wie bereits erwähnt besteht also zwischen den ein System beschreibenden Prozessen eine kooperative Wechselwirkung. Im Fall zweier kooperierender Prozesse Ξ_1, Ξ_2 stellt sich das folgendermaßen dar, dass zu einem Zeitpunkt $t \in S \subset T$ zwei Signalwerte $(\xi_1(t), \xi_2(t)) \in X_1 \times X_2$ mit $\xi_1 \in \Xi_1, \xi_2 \in \Xi_2$ eintreten können.

- die Menge

$$\Xi_K \in \mathsf{P}(T \times X_1 \times X_2) \tag{2.60}$$

heißt kooperativer Prozess mit den zweidimensionalen Signalen

$$\xi \in \Xi_K, \quad \xi : D(\xi) \subset T \mapsto X_1 \times X_2 \tag{2.61}$$

Mit dem zweidimensionalen Phasenraum $X_1 \times X_2$ aus (2.61) können die Überlegungen aus den Abschnitten 2.2.1 und 2.3 zur Vollständigkeit und temporalen Wechselwirkung auf kooperative Prozesse übertragen werden. An die Stelle von X tritt dann $X_1 \times X_2$.

Das zweidimensionale Signal aus (2.61) lässt sich auch als 2-Tupel schreiben:

$$\xi(t) = (\xi_1(t), \xi_2(t)) \tag{2.62}$$

Umgekehrt lassen sich die beiden Teilsignale ξ_1, ξ_2 aus (2.62) auch als Projektion von ξ interpretieren:

$$\xi_1 = pr_1 \circ \xi \quad \xi_2 = pr_2 \circ \xi \tag{2.63}$$

mit $\xi_i(t) = pr_i(\xi(t)) = pr_i(\xi)(t)$. Die Projektion pr_i und die Signalabbildung $\xi(t)$, mit der den Zeiten $t \in T$ die Werte aus $X_1 \times X_2$ zugeordnet werden, sind kommutativ. Also kann man die Teilprozesse Ξ_1, Ξ_2 als Projektion von Ξ auffassen.

- die Teilprozesse Ξ_1, Ξ_2 mit

$$\begin{aligned} \Xi_1 &= \{pr_1(\xi) \mid \xi \in \Xi_K\} \\ \Xi_2 &= \{pr_2(\xi) \mid \xi \in \Xi_K\} \end{aligned} \tag{2.64}$$

heißen Projektionsprozesse.

Der Kooperationsprozess Ξ_K lässt sich somit in einer zu (2.60) äquivalenten Darstellung als zweistellige Relation definieren:

$$\Xi'_K \subseteq \Xi_1 \times \Xi_2 \tag{2.65}$$

Der kooperative Prozess selbst stellt mit (2.65) eine Wechselwirkungsrelation dar. Ähnlich wie bei der temporalen Wechselwirkung in (2.32) gibt Ξ'_K ein qualitatives Maß für die Kooperation zwischen zwei Prozessen an. Im Grenzfall

$$\Xi'_K = \Xi_1 \times \Xi_2 \tag{2.66}$$

ist Ξ_1 mit Ξ_2 komplett verträglich. Die Prozesse sind dann unabhängig voneinander bzw. autonom. Es findet also keine Kooperation statt. Im anderen - durchaus üblichen - Grenzfall sind Ξ_1 und Ξ_2 durch Ξ'_K bijektiv einander zugeordnet. Jedem $\xi_1 \in \Xi_1$ ist mit genau einem $\xi_2 \in \Xi_2$ verträglich. Die Prozesse sind dann wechselseitig funktional.

Die kooperative Wechselwirkung in (2.65) zwischen zwei Teilprozessen ist insbesondere in den technischen Wissenschaften von besonderer Bedeutung. Dort heißen die Signale $\xi_1 \in \Xi_1$ Eingang (input), die $\xi_2 \in \Xi_2$ heißen Ausgang (output) des kooperativen Prozesses Ξ'_K. Wir führen mit Φ_1 und Φ_2 noch die entsprechenden kausalen Wechselwirkungsrelationen ein, die den Eingang auf den Ausgang abbilden bzw. umgekehrt

- Die Relationen

$$\begin{aligned} \Phi_1 &: \Xi_1 \mapsto \Xi_2, \quad \Phi_1(\xi_1) = \{\xi_2 \mid (\xi_1, \xi_2) \in \Xi'_K\} \subset \Xi_2 \\ \Phi_2 &: \Xi_2 \mapsto \Xi_1, \quad \Phi_2(\xi_2) = \{\xi_1 \mid (\xi_1, \xi_2) \in \Xi'_K\} \subset \Xi_1 \end{aligned} \tag{2.67}$$

  heißen kooperative Wechselwirkungsrelationen oder auch IO-Relationen.

Diese Relationen sind mit (2.64) und nur im zweidimensionalen Fall bitotal, müssen aber nicht rechtseindeutig sein und sind deshalb im allgemeinen Fall keine Abbildungen.

2.4.1.1 Multivariable Prozesse

Die kooperative Wechselwirkungsrelation tritt natürlich auch bei Mehrgrößensystemen auf. Das zweidimensionale Signal aus (2.62) wird nun auf den n-dimensionalen Phasenraum erweitert. Ein n-dimensionales Signal lässt sich wieder als n-Tupel schreiben:

$$\xi = (\xi_1, \ldots, \xi_n) \tag{2.68}$$

Der kooperative Prozess Ξ'_K aus (2.65) ist dann multivariabel:

$$\Xi'_K \subset \Xi_1 \times \ldots \times \Xi_n \tag{2.69}$$

Die kooperativen Wechselwirkungsrelationen können i.A. beliebige Teiltupel $\xi_{\underline{i}}, \xi_{\underline{j}}$ aus ξ als Ausgänge und Eingänge einander zuordnen:

$$\Phi_{\underline{i}} : \Xi_{\underline{i}} \mapsto \Xi_{\underline{j}} \tag{2.70}$$

mit $\underline{i} = (i_1, \ldots, i_K), \underline{j} = (j_1, \ldots, j_L)$ und $K, L \in \mathbb{N} \quad K, L < n$.

Die Kopplung einzelner Teilprozesse $\Xi_{\underline{i}}, \Xi_{\underline{j}}$ durch die Wechselwirkungsrelationen $\Phi_{\underline{i}}$ kann bei realen Prozessen sehr komplex werden. Für die Simulation und Analyse solcher

kooperativer Prozesse ist die mathematische Struktur der Wechselwirkungsrelationen (algebraische/differenzielle Gleichungen, bijektiv/injektiv) und inwieweit durch Umstellen der gegebenen Wechselwirkungsrelationen bestimmte Teiltupel $\xi_{\underline{i}_{out}}$ aus Teiltupeln $\xi_{\underline{i}_{in}}$ in m Schritten berechnet werden können, von besonderem Interesse (siehe hierzu auch 2.4.2.2).

$$\xi_{\underline{i}_{out}} = \Phi_m \circ \ldots \circ \Phi_1(\xi_{\underline{i}_{in}}) \tag{2.71}$$

2.4.2 Kausalität

In engem Zusammenhang mit der kooperativen Wechselwirkung steht der Begriff der Kausalität. Hierunter wird im Allgemeinen das Prinzip verstanden, dass die im Prozessverlauf später eintretenden Vorgänge keinen Einfluss auf die vorangegangenen ausüben können. Insofern ist dann das Temporalprinzip die notwendige Vorraussetzung für die Kausalität. Die eben erläuterte Begriffsbedeutung soll deshalb im weiteren als 'Kausalität im Sinne der Zeit' genannt werden. Dieser Sachverhalt wird auch als Kausalprinzip [1] bezeichnet [Schnieder 1993].

Neben dieser allgemein anerkannten Vorstellung von Kausalität ([Schnieder 1993], [Wunsch 2000], [Unbehauen 1997]) hat sich im Zusammenhang mit der Modellsimulation der Begriff der 'Berechnungskausalität' etabliert. Dieser Begriff zielt darauf ab, zu beschreiben, inwieweit durch ein Systemmodell die Ein- und Ausgangsgrößen und die zu deren Berechnung notwendigen kooperativen Kopplungsrelationen festgelegt werden, siehe z.B. [Borutzky 2000], [Gawthrop 1996]. Denn bestimmte Beschreibungsmittel ermöglichen die Systemmodellierung ohne Festlegung der Berechnungskausalität, z.B. Bondgraphen [Paynter 1961], [Ljung 1994] oder die objektorientierte Modellbeschreibungssprache Modelica [Modelica Association 2000]. Dieser Begriffsbedeutung soll 'Kausalität im Sinne der Kooperation' bezeichnet werden.

Beide Kausalbegriffe werden auf der Basis der kooperativen Wechselwirkungsrelation aus Gleichung (2.67) im folgenden näher erläutert.

2.4.2.1 Kausalität im Sinne der Zeit

Wie bereits erläutert, lässt sich Kausalität zunächst einmal bezüglich der Wechselwirkung von Zukunft und Vergangenheit interpretieren: zukünftige Ereignisse haben keinen Einfluß auf die Vergangenheit. Nach [Wunsch 2000] lässt sich dies auf die kooperative Wechselwirkungsrelation Φ_1 in (2.67) eines binären Prozesses übertragen.

- Die Wechselwirkungsrelation Φ_1 heißt nicht-antizipativ (kausal), wenn für alle $\xi_1, \xi_1' \in \Xi_1$ und für alle $\tau \in D(\Xi_1)$ gilt:

$$\xi_1 \overset{\tau}{\circ} \xi_1' \in \Xi_1 \quad \Rightarrow \quad \Phi_1(\xi_1) \mid \overline{T}^{\tau} = \Phi_1(\xi_1 \overset{\tau}{\circ} \xi_1') \mid \overline{T}^{\tau} \tag{2.72}$$

[1] Prominentes Gegenbeispiel aus der Politik wäre eine rückwirkende Steuersatzänderung, die in diesem Zusammenhang als nicht-kausal zu bezeichnen ist

Der Term $\Phi_1 \mid \overline{T}^\tau$ bedeutet mit (2.27) die Einschränkung von Φ_1 auf die Vergangenheit $\overline{T}^\tau$ von τ. Φ_1 ist also nicht-antizipativ, wenn sich die Vergangenheit von Φ_1 unabhängig von zukünftigen Verlauf des Eingangs ξ_1 verhält.

Für den binären kooperativen Prozess Ξ_K gilt nun:

- Der kooperative Prozess $\Xi_K \subset \Xi_1^* \times \Xi_2^*$ mit $\Xi_{1,2}^* = [T, X_{1,2}]$, vgl. auch (2.4), heißt kausal, wenn Φ_1, Φ_2 kausal (oder nicht-antizipativ) sind.

Diese Kausalitätsdefinition lässt sich auch auf multivariable Prozesse verallgemeinern. An die Stelle von $\Phi_{1,2}$ in der obigen Definition treten dann alle Φ_i des kooperativen Prozesses.

Eine ähnliche Kausalitätsdefinition findet sich in [Unbehauen 1997]. Dort heißt ein System kausal, wenn für dessen kooperative Wechselwirkungsrelationen folgendes gilt:

$$\bigwedge_{t \le \tau} \Big(\quad \xi_1(t) = \xi_1'(t) \quad \Rightarrow \quad \Phi_1(\xi(t)) = \Phi_1(\xi_1'(t)) \quad \Big) \tag{2.73}$$

Man kann zeigen, dass (2.72) und (2.73) äquivalent sind. Darauf wird hier verzichtet.

2.4.2.2 Kausalität im Sinne der Kooperation (Berechnungskausalität)

Multidimensionale, kooperative Prozessen werden im Allgemeinen durch komplexe Wechselwirkungsrelationen beschrieben, siehe (2.69) und (2.70), so dass a priori keine Aussage darüber gemacht werden kann, welche Signale ξ_i aus $\xi = (\xi_1, \ldots, \xi_i, \ldots, \xi_n)$ zu den Eingangs- und Ausgangsgrößen zu zählen sind. Die Wahl der Eingangs- und Ausgangsgrößen hängt davon ab, inwieweit die Wechselwirkungsrelationen umgeformt (z.B. invertiert) und so miteinander verknüpft werden können, dass die Ausgangsgrößen aus den Eingangsgrößen berechenbar sind. Bei Beschreibungsmitteln mit unidirektionalen Signalflüssen wird die Berechnungsreihenfolge für Ausgangs- und Zwischengrößen bereits beim Modellieren festgelegt. Diese Berechnungsreihenfolge nennt man auch Berechnungskausalität.

- Sind die Ein- und Ausgangsgrößen $\xi_{\underline{i}_{in}}, \xi_{\underline{i}_{out}}$ eines kooperativen Prozesses festgelegt und existiert eine Beschreibung in Form von kooperativen Wechselwirkungsrelationen $\Phi_1, \ldots, \Phi_m$, so dass gilt:

$$\xi_{\underline{i}_{out}} = \tilde{\Phi}(\xi_{\underline{i}_{in}}) \quad \text{mit} \quad \tilde{\Phi} = \Phi_m \circ \ldots \circ \Phi_1 \tag{2.74}$$

  dann heißt der Prozess kausal im Sinne der Kooperation bzw. berechnungskausal.

Die Ausgangsgrößen $\xi_{i_{out}}$ lassen sich also in m Zwischenschritten aus den Eingangsgrößen $\xi_{i_{in}}$ berechnen (vgl. (2.71)). Bei dynamischen Prozessen treten dann neben den Ein- und Ausgangsgrößen bis zu m Zwischengrößen auf, die in diesem Zusammenhang als Zustandsgrößen bezeichnet werden.

Kausale Prozessbeschreibungen liegen z.B. mit Signalflussdiagrammen oder Blockschaltbildern vor. Sie ermöglichen die Darstellung rückwirkungsfreier Wechselwirkungen. Rückwirkungen müssen explizit modelliert werden. Die Ausgangsgrößen können dann auch auf der rechten Seite von (2.74) stehen.

Im folgenden Beispiel in Abbildung 2.7 wird dies deutlich. Gegeben sei ein Blockschaltbild mit Rückführung.

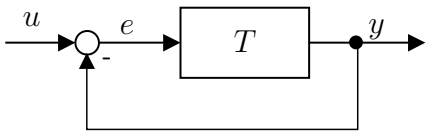


Abbildung 2.7: Blockschaltbild mit Rückführung

Es handelt sich um einen funktionalen, kooperativen Prozess $\Xi = \{u, e, y\}$. Die kooperativen Wechselwirkungsrelationen lauten:

$$\begin{aligned} \Phi_a(u, y) &\mapsto e \quad u - y = e \\ \Phi_b(e) &\mapsto y \quad T \cdot e = y \end{aligned} \tag{2.75}$$

Mit der Ausgangsgröße y und der Eingangsgröße u ergibt sich die zu (2.74) äquivalente Form:

$$y = \Phi_b \circ \Phi_a(u, y) = T \cdot (u - y) \tag{2.76}$$

Aufgrund der Rückführung steht die Ausgangsgröße y auf der rechten Seite von (2.76). Sofern es sich bei dem Block T um ein dynamisches System handelt, wird (2.76) zu einer Differenzial- oder Integralgleichung, die sich numerisch gut lösen lässt. Ist T ein algebraischer Block (*feedthrough*-Block), liegt mit (2.76) eine sogenannte 'algebraische Schleife' vor, die zwar analytisch durch Umstellen der Gleichung leicht gelöst werden kann, die aber bei der Behandlung mit numerischen Simulationssystemen Probleme bereitet. In MATLAB/SIMULINK z.B. wird der Ergebniswert y auf der Basis früherer Werte geschätzt und dann iterativ angepasst, bis der Schätzwert $\tilde{y}$ die Gleichung (2.76) erfüllt [THE MATH WORKS 1994].

Bei der Modellierung rückwirkungsbehafteter Systeme (z.B. mechanische, hydraulische oder elektrische Netzwerke) eignen sich jedoch besser Beschreibungsmittel mit bidirektionalen Signalflüssen, wie z.B. Bondgraphen [PAYNTER 1961], da hier mit den Modellen die Berechnungskausalität noch nicht festgelegt ist. Ausgangs- und Eingangsgrößen (und damit die Berechnungskausalität) können noch im Nachhinein für ein Modell festgelegt werden. Hierfür gibt es formale Methoden [BORUTZKY 2000].

2.5 Zustandsdarstellung

Im vorigen Abschnitt 2.3 wurde die temporale Wechselwirkung zwischen den Signalen ξ innerhalb eines Prozesses Ξ als wichtige Prozesseigenschaft erläutert. Hierzu wurde der Gedächtnisbegriff eingeführt (siehe (2.35)). Darauf aufbauend konnten dann Prozesse anhand ihrer Gedächtnislänge charakterisiert (siehe (2.37), (2.38) und (2.39)) und hierarchisch eingeordnet werden (siehe (2.40)).

Von besonderer Bedeutung sind nun in diesem Zusammenhang die Prozesse ohne Gedächtnis (bzw. mit Gedächtnislänge $L = 0$), die sogenannten Markov-Prozesse, siehe auch (2.38). Solche Markov-Prozesse ermöglichen es, mit genauer Kenntnis über die aktuelle Prozesssituation die Prozesszukunft vorherzusagen. Kenntnisse über die Prozessvergangenheit sind nicht weiter notwendig.

Wie sich leicht vorstellen lässt, hat diese temporale Eigenschaft der Markov-Prozesse natürlich auch Auswirkungen auf ihre Prozessstruktur. Denn aufgrund der Tatsache, dass immer von einer Phase $\xi(t_1)$ auf eine spätere Phase $\xi(t_2)$ mit $t_2 > t_1$ geschlossen werden kann, lassen sich Markov-Prozesse durch Phasenstrukturen (siehe (2.17)) mit zweistelligen Phasenrelationen (siehe (2.16)) beschreiben. Sie sind also immer durch einen einen $\underline{T}$-vollständige Prozess mit $\underline{T} \in T^2$ darstellbar:

- Ein beliebiger $\underline{T}$-vollständiger Markov-Prozess $\Xi \in \mathcal{X}_0$ lässt sich folgendermaßen darstellen:

$$\xi \in \Xi \quad \Leftrightarrow \quad \bigwedge_{\underline{t} \in \underline{T}} \left(\pi_{\underline{t}}(\xi) \in \rho(\underline{t}) \subset X^2 \right) \tag{2.77}$$

  wobei $\underline{t} = (t_1, t_2)$ geordnete Zeitpaare mit $t_1 < t_2$ sind.

Die Prozessdarstellung in (2.77) wird gemeinhin auch als Zustandsdarstellung bezeichnet. Die Phasen $\xi(t) = x \in X$ heißen Zustand, der Träger X heißt Zustandsraum, die zeitlichen Verläufe ξ heißen Zustandstrajektorien. Der den Prozess beschreibende Zustandsübergang $\pi_{\underline{t}}(\xi) = (\xi(t_1), (\xi(t_2))$ überführt den Vorzustand $\xi(t_1)$ bzw. Zustand ante in den Nachzustand $\xi(t_2)$ bzw. Zustand post, [Polke 2000]. Dieser elementare Prozess der Zustandstransformation ist Grundlage einer formalisierten Prozessbeschreibung, die in [GMA 2003] als VDI/VDE-Norm vorliegt (siehe Abschnitt 4.4) und bereits in [Lauber 1996] in Grundzügen vorgestellt wurde.

Der Zustandsbegriff ist vorallem in der Technik und Naturwissenschaft von großer praktischer und theoretischer Bedeutung und konnte dort in vielfacher Weise formalisiert werden ([Wunsch 1985]). In der Informatik spielen die Zustandsautomaten eine große Rolle. Die dort beschriebenen Markov-Prozesse sind meist irreversibel und nicht-determiniert, der Zustandsraum meist algebraisch unstrukturiert. Zur Beschreibung nicht-autonomer gesteuerter Systeme mit dem Eingabealphabet X, dem Zustandsraum Z und dem Ausgabealphabet Y dienen die Mealy-Automaten ([Wunsch 1985],

[Bronstein 1995b]).

$$\begin{aligned} f &: Z \times X \mapsto Z \quad z(t_{i+1}) = f(z(t_i), x(t_i)) \\ g &: Z \times X \mapsto Y \quad y(t_i) = g(z(t_i), x(t_i)) \end{aligned} \tag{2.78}$$

mit $t_i \in \mathbb{Z}$.

Jedoch hat der Zustandsbegriff auch weitreichend Einzug gehalten in die klassische Mechanik und die Regelungstechnik. Für die dort im 'Kontinuierlichen' stattfindenen Prozesse, also für reelle Träger $T, X \subseteq \mathbb{R}$, gilt folgende Zustandsdarstellung:

$$\begin{aligned} \dot{z}(t) &= f(z(t), x(t)) \\ y(t) &= g(z(t), x(t)) \end{aligned} \tag{2.79}$$

Die Analogie von (2.79) zu (2.77) bzw. (2.78) ist sofort ersichtlich, denn aus (2.78) folgt:

$$z(t_{i+1}) - z(t_i) = f(z(t_i), x(t_i)) - z(t_i) \equiv f'(z(t_i), x(t_i)) \tag{2.80}$$

Mit $z(t_{i+t}) - z(t_i) = \Delta z$ und $t_{i+1} - t_i = \Delta t = 1 = const$ kann (2.80) folgendermaßen umgeformt werden:

$$\frac{\Delta z}{\Delta t} = f'(z(t_i), x(t_i)) \tag{2.81}$$

Unter der Annahme reeller Träger in (2.81) kann der Grenzübergang

$$\lim_{\Delta t \to 0} \frac{\Delta z}{\Delta t} = \dot{z}(t) \tag{2.82}$$

durchgeführt werden. (2.81) wird dann mit (2.82) zu (2.79).

Besonders elegant zeigt sich die Analogie zwischen (2.78) und (2.79) mit den mathematischen Mitteln des Booleschen Differenzialkalküls, [Bochmann 1981]. Das Boolesche Differenzialkalkül ermöglicht es, mit Differenzialen für mehrdimensionale, binäre Variablen $x \in \mathbb{B}^k$ zu rechnen. $\mathbb{B}^k$ ist ein linearer Vektorraum mit den Koordinaten $x = (x_1, \cdots, x_k)$ mit $x_i \in \mathbb{B}$. Das Differenzial $\mathrm{d}x$ ist dann folgendermaßen defininiert:

$$\mathrm{d}x = x_0 \oplus x \tag{2.83}$$

Der Operator $\oplus$ entspricht dem exklusiven ODER bzw. XOR. Das Differenzial $\mathrm{d}x$ ist die erforderliche Änderung, um x aus x_0 zu erreichen.

Mit dem Differenzial aus (2.83) kann man folgendermaßen die Zustandsgleichung für binäre Systeme formulieren, [Bochmann 2000]: Ausgehend von (2.79) erfolgt zunächst

wieder der Übergang auf diskrete bzw. ereignisdiskrete[2] Prozessträger $X \subset \mathbb{B}^k, T \subset \mathbb{Z}$. Dann wird, wie in (2.81) und (2.82) bereits gezeigt, aus dem kontinuierlichen Differenzialquotienten ein diskretes Differenzial dz, das direkt aus (2.83) entnommen werden kann:

$$\dot{z} = \frac{\mathrm{d}z}{\mathrm{d}t} \underbrace{=}_{\mathrm{d}t \to \Delta t = 1} \mathrm{d}z = z(t) \oplus z(t+1) \tag{2.84}$$

Somit gilt (2.79) auch für Markov-Prozesse mit binären/ereignisdiskreten Trägermengen. Aber im Gegensatz zur herkömmlichen Definitionen von Zustandsautomaten wie in (2.78) wird durch (2.84) nicht der Folgezustand eines Prozesses, sondern das Differenzial zwischen Ist- und Folgezustand (also eine Zustandsänderung) bestimmmt.

Die Spezifikation diskreter Prozesse durch boolesche Differenziale lässt sich anschaulich anhand des in der folgenden Abbildung 2.8 gezeigten Petrinetzes (siehe hierzu Abschnitt 4.1) darstellen. Die Transition t_1 ist mit der Transition t_2 über eine Stelle verbunden, deren Markenbelegung dem Zustand $z(t_i)$ entspricht.

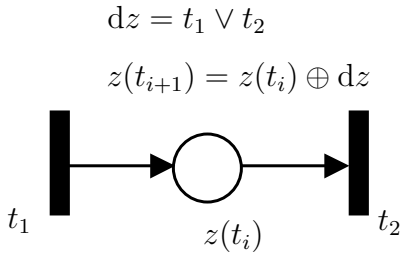


Abbildung 2.8: Boolesches Differenzial an einem Petrinetz

Sobald die Transitionen t_1 oder t_2 schalten, tritt eine Zustandsänderung $\mathrm{d}z$ ein, wobei die Schaltfähigkeit der Transitionen vom aktuellen Zustand $z(t_i)$ abhängig ist. Der Folgezustand $z(t_{i+1})$ ergibt sich gemäß (2.83) aus der XOR-Verknüpfung vom aktuellen Zustand $z(t_i)$ und dem booleschen Differenzial $\mathrm{d}z$. Dieses Verahren lässt sich leicht mit gängigen numerischen Simulationswerkzeugen wie z.B. MATLAB oder OCTAVE implementieren und ist Basis für die diskrete Zustandsdarstellung in der am Ende der Arbeit vorgestellten Werkzeugumgebung zur Simulation kombinierter Bondgraphen/Petrinetz-Modelle.

Wie bereits erwähnt ist die Zustandsdarstellung in der Systemtheorie und der Informatik weit verbreitet. Dies liegt daran, dass sich viele Eigenschaften von Prozessen nur oder besonders gut in Zustandsform analysieren lassen. Erwähnt seien hier z.B. Beobachtbarkeit/Steuerbarkeit in der Regelungstechnik ([LUNZE 2002a], [MÜLLER 1997]) oder Erreichbarkeit und Verifikation mittels *model checking* in der Informatik ([BERARD 2001]).

[2] Bei ereignisdiskreten Systemen kann die Annahme getroffen werden, dass es sich bei Ereignissen um eine abzählbare, geordnete Menge handelt, die auf einen diskreten Zeitbereich $T \subset \mathbb{Z}$ abgebildet werden kann.

Desweiteren kann man Prozesse in Zustandsform gut simulieren, da es hierfür viele analytische und numerische Lösungsverfahren gibt. Abschließend soll noch darauf aufmerksam gemacht werden, dass in [Wunsch 2000] gezeigt wird, dass sich jeder vollständige Prozess als Markov-Prozess darstellen lässt. Somit hat die Zustandsdarstellung auch universellen Charakter für die Prozesstheorie.

2.6 Hybride Systeme

Die Beschreibung technischer und naturwissenschaftlicher Phänomene mittels Markov-Prozessen (Zustandsdarstellung) hat sich in den Ingenieurwissenschaften und Informatik fest etabliert. Während die Markov-Prozesse in der Informatik weitestgehend auf algebraisch unstrukutrierten Trägermengen (allenfalls mit einem geordneten, abzählbaren Ereignisraum) aufbauen und explizit mit zweistelligen Phasenrelationen formuliert werden, sind die Trägermengen physikalisch-technischer Markov-Prozesse meist reell, ihre Trajektorien sind die Lösungen von prozessdefinierenden Differenzialgleichungen. Wie bereits gezeigt lassen sich die in Informatik und Ingenieurwissenschaften gebräuchlichen Zustandsdarstellungen ineinander überführen, doch bedingen die unterschiedlichen mathematischen Eigenschaften der Trägermengen (und damit auch der Prozessdarstellungen) derart unterschiedliche Analysemethoden, dass sich ganz eigene, trägermengenspezifische Prozesssichten entwickelt haben. Deshalb werden die diskreten Systeme (im Sinne von ereignisdiskret) von den kontinuierlichen Systemen (im Sinne von zeitkontinuierlich) unterschieden.

Die strikte Trennung von diskreten und kontinuierlichen Systemem hat sich jedoch in der Automatisierungstechnik aufgrund des Einsatzes digitaler Rechentechnik als nachteilig erwiesen, da viele Systeme, bestehend aus kontinuierlicher Strecke und diskreter Steuerung, nicht getrennt betrachtet werden dürfen. Daher sind in jüngerer Zeit die Ansätze zur Modellierung und Analyse sogenannter 'hybrider', gemischter diskret-kontinuierlicher Systeme verstärkt in den Blickpunkt automatisierungstechnischer Forschung gelangt[3].

Auf die besonderen Probleme bei der Beschreibung und Modellbildung verfahrenstechnischer Prozesse weist Buchner in [Buchner 1998] hin. Die in der Verfahrenstechnik vorkommenden Prozesse zeichen sich häufig durch qualitative, unscharfe Prozesseigenschaften aus und können nur auf der Basis nicht-metrischer Eigenschaftsskalen wie z.B. Nominal- oder Ordinalskalen geeignet beschrieben werden. Bei diesen Skalen handelt es sich i.A. um abzählbare Mengen, die im Fall der Nominalskala keinerlei algebraische Struktur aufweisen und die im Falle der Ordinalskala höchstens geordnet sind. Aufbauend auf diesen Überlegungen stellt Buchner eine Anforderungsliste für Beschreibungsmittel vor, die zur Beschreibung solcher Prozesse mit nicht-metrischen Eigenschaften geeignet sind.

Wir werden im folgenden einen prozesstheoretischen Ansatz zur Definition des Be-

[3] vgl. z.B. das von der DFG geförderte Schwerpunktprogramm KONDISK, [Engell et al. 2002]

griffs 'hybrides System' vorstellen und danach den Zusammenhang zwischen hybriden Systemen/Prozessen und der temporalen Wechselwirkung erläutern.

Wie bereits erwähnt beruht in der Automatisierungstechnik das Verständnis von hybriden Systemen und Prozessen auf deren Zusammensetzung aus kontinuierlichen und diskreten Teilprozessen. Das folgende Zitat macht dies deutlich:

'Die ... Systeme, die dadurch gekennzeichnet sind, dass die Wechselwirkung kontinuierlicher und diskreter Prozesse ihr Verhalten entscheidend bestimmt, werden als hybride dynamischen Systeme bezeichnet'[4]. Eine ähnliche Definition findet sich z.B. auch in [Nenninger 2001].

2.6.1 Kontinuierliche und Diskrete Systeme

Hybride Prozesse sind damit kooperative Prozesse, wobei die beteiligten Teilprozesse diskret (Ξ_d) und kontinuierlich (Ξ_k) sind. Die Kooperation zwischen Prozessen lässt sich durch eine kooperative Wechselwirkung repräsentieren, siehe (2.65). Damit ergibt sich für einen hybriden Prozess die folgende (dem hybriden Prozesse äquivalente) kooperative Wechselwirkung:

$$\Xi'_h \subseteq \Xi_k \times \Xi_d \tag{2.85}$$

Die Definition hybrider Prozesse beruht also auf dem Begriffspaar *kontinuierlich* und *diskret*. Was aber unterscheidet kontinuierliche von diskreten Prozessen? Überlicherweise unterscheidet man diese anhand ihrer Trägermengen. Kontinuierliche System haben einen reellen Phasenraum und Zeitbereich:

$$\Xi \in \mathcal{X}_k \Leftrightarrow X \subseteq \mathbb{R} \wedge T \subseteq \mathbb{R} \tag{2.86}$$

Diese Trägermengen sind überabzahlbar und ermöglichen die Definition der kontinuierlichen Prozesse mit Differenzialgleichungen.

Die Trägermengen diskreter Prozesse können sehr unterschiedlich sein. Ihr wichtigstes Merkmal ist die Abzählbarkeit. In vielen Fällen sind die Träger auch noch endlich.

Probleme bereiten hier die Abtastprozesse (zeitdiskret) bzw. die digitalen Prozesse (zeit- und wertdiskret), deren Träger folgende Gestalt haben.

$$\begin{aligned} \Xi \in \mathcal{X}_{Abtast} &\Leftrightarrow T \subseteq \{t | t = n \cdot T_S\} \quad (X \subseteq \mathbb{R}) \\ \Xi \in \mathcal{X}_{Digital} &\Leftrightarrow T \subseteq \{t | t = n \cdot T_S\} \quad \wedge \quad X \subseteq \{x | x = n \cdot A\} \end{aligned} \tag{2.87}$$

Die Periode T_S heißt *Abtast- oder Samplerate*, A heißt *Auflösung*. Diese Prozesse können mit Differenzengleichungen beschrieben werden und haben eine wichtige Bedeutung

[4] [Engell 1997], S. 152

für technische Systeme, die mit Rechnerunterstützung automatisiert werden. Ihre Träger sind ganz offensichtlich abzählbar. Genügend hohen Abtastzeiten und Auflösungen vorrausgesetzt, werden diese Prozesse trotzdem zu den kontinuierlichen gezählt[5].

Die Abzählbarkeit der Trägermängen ist somit kein stichhaltiges Kriterium. Auch die Ordnung der Trägermengen bietet kein exklusive Unterscheidungsmerkmal. Zwar sind die Träger kontinuierlicher System i.A. geordnet, doch trifft das Gegenteil nicht für alle diskreten Prozesse zu. Ihre Träger können, müssen aber nicht ungeordnet sein.

Einen wichtigen Hinweis auf die Unterscheidung kontinuierlicher/diskreter Systeme wird in [Lunze 2002b] gegeben. Hier wird das Lipschitz-Kriterium[6] zur Definition hybrider Systeme herangezogen. *Hybrid phenomena are state transitions that cannot be represented or analysed by the methods developed in continuous or discrete systems theory. That is, these phenomena do not satisfy the Lipschitz condition ...*[7]

Nach Lunze zeichnen sich hybride Systeme dadurch aus, dass ihre Zustandstrajektorien nicht stetig sind. Lunze bezeichnet diese Unstetigkeiten *'Zustandssprünge'* oder *'state jumps'*. Aufgrund dieser Unstetigkeiten entstehen in der Zustandsdifferenzialgleichung Unendlichkeitstellen, die mathematisch durch Dirac-Impulse dargestellt werden können. Aufgrund der Dirac-Impulse erüllt diese Zustandsdifferenzialgleichung nicht die Lipschitz-Bedingung. Eine hybride Differenzialgleichung (2.88) besteht dann aus einem Lipschitz-kontinuierlichen Teil $\overset{\circ}{f}$, dem systemabhängige Dirac-Impulse $f_i \cdot \delta\big(g_i(x,u,t)\big)$ überlagert sind.

$$\dot{x} = f(x,u,t) \quad \text{mit} \quad f(x,u,t) = \overset{\circ}{f}(x,u,t) + \sum_{i=1} f_i \cdot \delta\big(g_i(x,u,t)\big) \tag{2.88}$$

Die Sprünge treten dann zu den Zeiten t_k auf, an denen die Funktion $g_i(x,u,t_k) = 0$ wird.

Es gibt jedoch verschiedene Differenzialgleichungen, die zwar die Lipschitz-Bedingung nicht erfüllen, aber keine Zustandssprünge aufweisen. Prominentes Beispiel ist die folgende Differenzialgleichung (2.89):

$$\dot{\xi} = 3 \cdot \xi^{\frac{2}{3}} \tag{2.89}$$

Die Lösung dieser Differenzialgleichung sind unendlich viele Parabeln der Form $\xi = (t-c)^3$ und die x-Achse $\xi = 0$. Ferner sind alle Trajektorien, die aus einem Stück der x-Achse bestehen und von kubischen Parabelbögen umrahmt sind, eine Lösung von (2.89).

In der folgenden Abbildung 2.9 sind vier dieser Lösungen gezeigt. Neben der kubische Parabel durch den Nullpunkt (1) und der x-Achse (2) sind dies die beiden Trajektorien, die links bzw. rechts vom Nullpunkt aus der x-Achse bestehen und mit einem Parabelast vervollständigt werden (3,4).

[5] man spricht auch von 'quasikontinuierlichen' Prozessen bzw. Systemen, siehe [Engell 1997]
[6] siehe z.B. [Collatz 1949], S. 28 ff
[7] [Lunze 2002b], S. 5

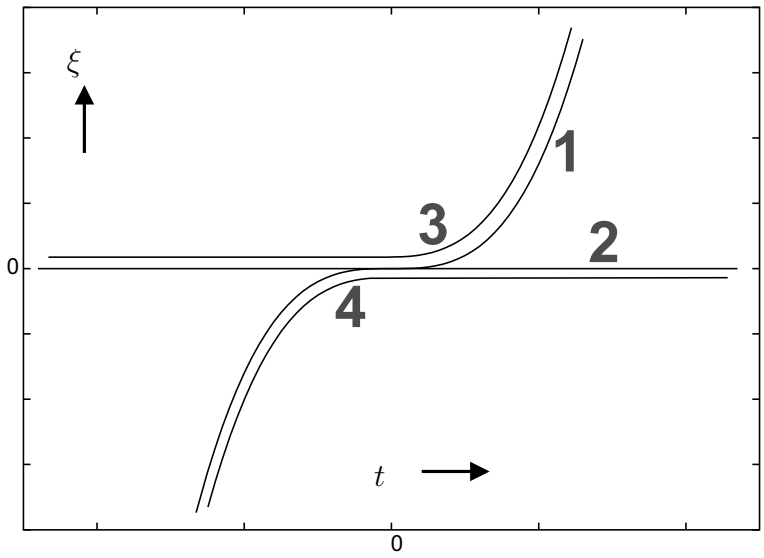


Abbildung 2.9: Nicht-determinierte, irreversible Differenzialgleichung

Trotz Verletzung des Lipschitz-Kriteriums hat der Prozess von (2.89) also Signale mit kontinuierlichem Verlauf. Entscheidend ist hier, dass der Prozess nicht-determiniert und irreversibel und damit nicht-kontinuierlich im klassischen Sinne ist: am Nullpunkt ist nicht klar, woher die Trajektorie kommt und wohin sie verläuft. Nach dem Eindeutigkeits-Satz der Differenzialrechnung[8] hängen diese Eigenschaften direkt mit dem Lipschitz-Kriterium zusammen. Denn nur Prozesse, deren bestimmende DGLen nicht eindeutig lösbar sind, können nicht-determiniert bzw. irreversibel sein. Somit führt das Lipschitz-Kriterium aufgrund des Eindeutigkeitssatzes auf eine allgemeinere Prozesscharakterisierung:

- kontinuierliche Prozesse Ξ_k haben meist überabzählbare, geordnete Trägermengen und sind funktional (reversibel, determiniert).
- diskrete Prozesse Ξ_k haben immer abzählbare und meist geordnete Trägermengen und können irreversibel bzw. nicht-determiniert sein.

Gerade die Möglichkeit, irreversible und nicht-determinierte Prozesse beschreiben zu können, unterscheidet maßgeblich die Theorien der diskreten Systemen von denen kontinuierlicher Systeme. Wie man also sieht, ist eine strenge Einteilung in kontinuierliche und diskrete Prozesse allein aufgrund von mathematischen Eigenschaften der Trägermengen (Abzählbarkeit, Ordnung usw.) nicht möglich. Die temporale Wechselwirkung bietet hier ein wichtiges, unterstützendes Unterscheidungskriterium.

In der folgenden Abbildung 2.10 sind die wichtigsten Kriterium für die Unterscheidung kontinuierlicher und diskreter Systeme nochmals tabellarisch zusammengefasst.

[8] siehe z.B. [Collatz 1949], S. 34

	diskret	*kontinuierlich*
Trägermenge	abzählbar, geordnet & ungeordnet	abzählbar & überabzählbar, geordnet
temporale Wechselwirkung	funktional & nicht-funktional	funktional (reversibel, determiniert)
	Ξ_d	Ξ_k

Abbildung 2.10: Unterscheidungsmerkmale kontinuierlicher und diskreter Systeme

2.6.2 Kooperative und temporale Wechselwirkungen in hybriden Prozessen

Der rein kooperativen Wechselwirkung in (2.85) ist bei dynamischen Systemen immer auch eine temporale Wechselwirkung (Gedächtnissprozesse, Abschnitt 2.3.2) überlagert. Diese Überlagerung von Wechselwirkungen kann man mit Hilfe eines hybriden Markov-Prozesses darstellen, siehe (2.90) und (2.91). Grundlage für diese Darstellung ist der Mealy-Automat in (2.78), so dass es sich bei (2.91) um einen hybriden Automaten handelt.

$$\Xi_h = \{\xi | \xi = (\xi_k, \xi_d)\} \quad \xi_k \in \Xi_k \quad \xi_d \in \Xi_d \tag{2.90}$$

mit

$$\begin{aligned} \xi_k(t_{i+1}) &= f_1\big(\xi_k(t_i), \xi_d(t_i), u_k(t_i)\big) \\ \xi_d(t_{i+1}) &= f_2\big(\xi_d(t_i), \xi_k(t_i), u_d(t_i)\big) \end{aligned} \tag{2.91}$$

Der kontinuierliche Zustand $\xi_k(t_{i+1})$ hängt also nicht nur vom vorigen Zustand $\xi_k(t_i)$, sondern auch vom diskreten Zustand $\xi_d(t_i)$ ab. Gleiches gilt für den diskreten Zustand $\xi_d(t_{i+1})$. In (2.91) finden auch noch die kontinuierlichen bzw. diskreten Eingangsgrößen u_k und u_d Berücksichtigung. Der hybride Automat in (2.91) ist die Grundlage für die in Kapitel 6 beschriebene Simulationsumgebung.

Für die Zeiten t_i, t_{i+1} gilt

$$t_{i+1} > t_i \quad \wedge \quad t_{i+1}, t_i \in D(\Xi_k) = \bigcap_{\xi_k \in \Xi_k} D(\xi_k) \tag{2.92}$$

Da bei hybriden System die Definitionsbereiche von kontinuierlichen und diskreten Teilprozessen $D(\Xi_k)$, $D(\Xi_d)$ nicht gleich sind

$$\bigcap_{\xi_k \in \Xi_k} D(\xi_k) = D(\Xi_k) \neq D(\Xi_d) = \bigcap_{\xi_d \in \Xi_d} \tag{2.93}$$

muss es eine Funktion μ mit

$$\mu : D(\Xi_k) \mapsto D(\Xi_d) \tag{2.94}$$

geben, damit in (2.91) die diskreten Trajektorien $\xi_d(t_i)$, $\xi_d(t_{i+1})$ für alle $t_i, t_{i+1} \in D(\Xi_k)$ definiert sind. Solche Funktionen μ bezeichnet man in der Technik auch als *Halteglieder*.

Eine weitere wichtige Eigenschaft von f_1 und f_2 in (2.91) ist, dass sie die wechselseitige Abbildung von diskreten und kontinuierlichen Trägermengen beinhalten:

$$\begin{aligned} f_1 &: X_k \times X_d \mapsto X_k \\ f_2 &: X_d \times X_k \mapsto X_d \end{aligned} \tag{2.95}$$

Nach [Lunze 2002b] kann man diese Abbildungen in separaten Funktionen konzentrieren, die er im Falle von f_1 *Injector* und im Falle von f_2 *Quantiser* nennt. Neben der Prozesstheorie finden sich solche Funktionen auch in der Signalverarbeitung und sind dort die Grundlage von Analog/Digital- bzw. Digital/Analog-Wandlerschaltungen.

2.7 Zusammenfassung

In diesem Kapitel wurden grundlegende prozesstheoretische Begriffe eingeführt, die es ermöglichen, dynamische Phänomene formal zu charakterisieren.

Zu Beginn wurde gezeigt, wie mit n-stelligen Phasenrelationen Prozesse im Zeitbereich vollständig dargestellt werden können, und wie sich Gedächtnissprozesse beschreiben lassen.

Die danach erläuterte temporale Wechselwirkung ist eine Prozesseigenschaft, mit der die Begriffe 'determiniert' und 'reversibel' auf Basis folgender Fragestellung definiert werden können: inwiefern lässt sich von einem über eine bestimmten Zeitintervall bekannten Prozessverlauf auf die Zukunft (determiniert) bzw. auf die Vergangenheit (reversibel) des Prozesses schließen?

Die kooperative Wechselwirkung hingegen ist ein Maß für die Kooperation zwischen Prozessen. Neben der klassischen Kausalität im Sinne der Zeit existiert auch eine Kausalität im Sinne der Kooperation (Berechnungskausalität), die angibt, ob und inwieweit zwei Prozesse über Ursache-Wirkungsprinzipien miteinander verkoppelt sind.

Ein weiteres wichtiges Prinzip der Prozesstheorie ist die Zustandsdarstellung. Es wurde gezeigt, dass dieses Darstellungsprinzip gleichermaßen auf Prozesse mit algebraisch strukturierten (Systemtheorie) und Prozesse mit unstrukturierten Trägermengen (Informatik) anwendbar ist. Gerade deshalb ist die klassische Einteilung in diskrete und kontinuierliche Prozesse, bei der ja allein aufgrund der Trägermengen unterschieden wird, wenig sinnvoll. Es wurde deshalb abschließend gezeigt, dass das Begriffspaar Reversibilität/Determiniertheit hierbei ein wichtiges Unterscheidungkriterium darstellt und auch für die Definition hybrider Prozesse herangezogen werden kann.

3 Konzepte der Informationsmodellierung

Gegenstand des folgenden Kapitels ist die Untersuchung ausgewählter Konzepte zur Gewinnung, Strukturierung, Formalisierung und ggf. rechnerinterpretierbaren Repräsentation von Anwendungswissen. Diese Konzepte sind Teil der 'Wissensverarbeitung', die in [Schneider 1997] als Teildisziplin der Informatik definiert wird. Die auf Basis dieser Konzepte generierten Modelle, die zur Strukturierung bzw. Repräsentation von Anwendungswissens dienen, werden gemeinhin als Informationsmodelle bezeichnet. Zwei der Konzepte zur Informationsmodellierung bilden den Schwerpunkt des nun folgenden Kapitels: Ontologien und Objektorientierung. Sie wurden gewählt, da es sich dabei um gut erforschte und dokumentierte Konzepte handelt, die zur Repräsentation der hier vorgestellten Informationsmodelle für die Automatisierungstechnik prinzipiell in Frage kommen. Vorher jedoch sollen noch einige Bereiche der Wissensverarbeitung kurz vorgestellt werden.

Grundlegende Impulse erhielt die Wissensverarbeitung von der Forschung im Bereich Künstlicher Intelligenz (KI). Genesereth und Nilsson behaupten '*..., dass sich die KI hauptsächlich mit dem Problem der Repräsentation und des Gebrauchs von deklarativem [...] Wissen befasst.*'[1]. Deklaratives Wissen ist in diesem Zusammenhang explizites Wissen, das anwendungsunabhängig mit deklarativen Aussagen beschrieben wird. Für Genesereth und Nilsson stellt die Prädikatenlogik die Grundlage der Wissensrepräsentation dar. Auf die Formalisierung von Wissen durch prädikatenlogische Ausdrücke wird in Abschnitt 3.1 noch näher eingegangen.

Neben der grundlagenorientierten Forschung im Bereich Wissensverarbeitung haben sich insbesondere auch anwendungsbezogene Forschungszweige etabliert. In den Wirtschaftswissenschaften und der Wirtschaftsinformatik hat sich der Begriff 'Wissensmanagement' etabliert, der die organisatorischen und betriebswirtschaftlichen Aspekte von Wissensverarbeitung miteinbezieht. Stellvertretend für die in der einschlägigen Literatur üblichen Definitionen von Wissensmanagement soll hier [Voss 2001] zitiert werden: '*Wissensmanagement beschäftigt sich mit der wirtschaftlichen Unterstützung (im Sinne einer Strukturierung) der Generierung, Distribution und Allokation von Wissen.*'[2]. Wissensmanagement ist auch Gegenstand soziologischer Forschungsarbeit. In [Willke 1998] wird Wissensmanagement in den Kontext einer systemischen Soziologie gestellt, wobei hier insbesondere die menschlichen und sozialen Einflussfaktoren von Wissensmanage-

[1] [Genesereth 1989], S. VIII
[2] [Voss 2001], S. 317

ment behandelt werden: '*Wissensmanagement meint die Gesamtheit organisatorischer Strategien zur Schaffung einer intelligenten Organisation ... hinsichtlich der technologischen Infrastruktur geht es vor allem darum, ob, wie und wie effizient die Organisation eine zur ihrer Kooperationsweise kongeniale Kommunikations- und Infrastruktur nutzt.*'[3]

Es gibt eine Vielzahl von Anwendungsfeldern, die sich auf vielversprechende Weise der Wissensverarbeitungskonzepte bedienen (z.B. Medizin und Biologie). Auch in der Automatisierungstechnik existieren unterschiedliche Ansätze zur Nutzbarmachung von Wissensmanagement-Methoden. In [Schnieder 2001] z.B. wird gezeigt, welchen Nutzen grundlegende axiomatische Begriffsbestimmungen für das Engineering von Automatisierungssystemen haben und wie diese als Begriffsmodelle formalisiert werden können.

3.1 Ontologien

Ausgehend von der KI-Forschung kann Wissen als zentraler Gegenstand der Wissensverarbeitung von unterschiedlichen Standpunkten aus betrachtet werden [Guarino 1995a]: Wissen im funktionalen Sinn ist eine Eigenschaft, die einem rational agierenden Subjekt - in der KI-Forschung gemeinhin Agent genannt - zugeschrieben werden kann. Wissen ermöglicht dem Agenten, ein bestimmtes Ziel zu erreichen und ist damit zweckgebunden bzw. funktional. In einem zweckungebundenen Sinn ist Wissen das Ergebnis einer konzeptionellen Analyse über einen gegebenen Gegenstandsbereich und insofern mit Modellen repräsentierbar. Die modellhaften Ergebnisse dieser konzeptionellen Analyse werden im Bereich der Wissensverarbeitung auch als *Ontologien* bezeichnet.

Der *Ontologie*-Begriff stammt ursprünglich aus der Philosophie. Ontologie bezeichnet dort ' *... die Lehre vom Sein und seinem Aufbau.*'[4] und beschäftigt sich mit der Strukturierung und dem Erwerb menschlichen Wissens. Den Grundstein zu dieser Seins-Lehre hat Aristoteles mit 'Kategorien', der ersten seiner vier logischen Schriften, gelegt (siehe z.B. die aktuelle Reclam-Ausgabe [Rath 1998]). Darin definiert Aristoteles zehn Kategorien[5], die zur Strukturierung des Seienden dienen. Erstaunlich viele Konzepte der modernen Informationsmodellierung finden sich bereits in dieser Schrift. An dieser Stelle sei nur die Untergliederung in ein primäres und ein sekundäres Wesen (vergleichbar mit dem Klassen/Instanzen-Konzept der Objektorientierung) und die Vererbung von Wesenseigenschaften genannt.

Es soll an dieser Stelle nicht verschwiegen werden, dass der grundsätzlich philosophisch geprägte Begriff *Ontologie* natürlich nicht ohne Doppeldeutigkeit im Informatik-Kontext 'wiederverwendet' werden kann: eine ganze Wissenschaftsdisziplin dient hier als Namensgeber für das Ergebnis informatischer Analyseprozesse und muss sich sogar gefallen lassen, im Plural erwähnt zu werden. Das Entlehnen von Begriffen aus anderen Wissenschaften ist eine zweifelhafte Praxis in der Informatik. Darin zeigt sich eine

[3] [Willke 1998], S. 39
[4] [Hegenbart 1984]
[5] *Wesen*, *Quantität*, *Qualität*, *Relation*, *Ort*, *Zeit*, *Lage*, *Haben*, *Handeln*, *Widerfahren*

gewisse Hilf- und Sorglosigkeit bei der Verwendung von Begriffsbezeichnungen und der Bestimmung von Begriffsbedeutungen. Dies führt dann zwangsläufig zu so sinnbeugenden und sprachverwirrenden Begriffen wie 'Ontologien'.

Trotz allem verbergen sich hinter dem, was die Informatik mit 'Ontologien' meint, brauchbare Konzepte der Wissensverarbeitung, die sich für die Spezifikation der Strukturverträglichkeit als sehr geeignet erwiesen haben. Deshalb widmet sich das folgende Kapitel dem informatisch geprägten Ontologie-Begriff - mit Skepsis und unter den erwähnten Vorbehalten.

3.1.1 Formale Ontologiedefinitionen

In der Literatur zur Wissensverarbeitung lässt sich keine einheitliche Sichtweise auf den Begriff Ontologie erkennen, eine allgemeine Definition ist dementsprechend schwierig. Im Gegensatz zu den vielen informalen und eher unscharf formulierten Ontologie-Definitionen[6], wird im folgenden der Schwerpunkt auf formale Definitions-Ansätze gelegt.

Vielzitiert ist die folgende Definition von Gruber: '*An ontology is an explicit specification of a conceptualization.*'[7] Gruber bezieht sich in seiner Definition auf den Begriff *Konzeptualisierung* von Genesereth und Nilsson [Genesereth 1989]. Eine Konzeptualisierung besteht demnach aus einer Menge D von Objekten aus einem gegebenen Gegenstandsbereich, genannt Diskurswelt, und einer Menge R von n-stelligen Relationen über D.[8]

- Das 2-Tupel K mit

$$K = (D, R) \tag{3.1}$$

heißt Konzeptualisierung. D ist eine Menge von Objekten eines Gegenstandsbereiches und heißt Diskurswelt. R ist eine Menge n-stelliger Relationen $\rho \subset D^n$ bzw. $\rho \in \mathsf{P}(D^n)$. Für R gilt also:

$$R = \{\rho \mid \rho \in R^* = \bigcup_{n \in \mathbb{N}} \mathsf{P}(D^n)\}$$

Ausgehend von einer Konzeptualisierung K kann deklaratives Wissen über einen Gegenstandsbereich axiomatisch mit wahren, prädikatenlogischen Ausdrücken formalisiert werden. Dies soll kurz in einem Beispiel erläutert werden:

[6] als Beispiel möge hier eine Definition aus [Staab 2002], S. 200 dienen: 'Ontologien sind formale Modelle einer Anwendungsdomäne, die dazu dienen, den Austausch und das Teilen von Wissen zu erleichtern'

[7] [Gruber 1995], S. 908

[8] In [Genesereth 1989] sind noch Funktionen Teil dieser Konzeptualisierungs-Definition. Diese werden hier der Einfachheit halber unter Relationen subsumiert.

Beispiel

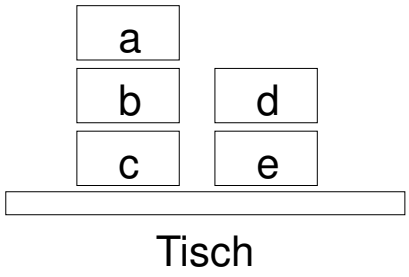


Abbildung 3.1: Beispiel einer Konzeptualisierung

Die Abbildung 3.1 zeigt fünf verschiedene Blöcke, die in der gezeigten Konstellation auf einem Tisch stehen.

Eine formale Konzeptualisierung könnte folgendermaßen lauten:

$$\begin{aligned} &K = (D, R) \quad \text{mit} \quad D = \{a, b, c, d, e\} \quad R = \{\rho_1, \rho_2, \rho_3\} \\ &\rho_1 = \{(a, b), (b, c), (d, e)\} \subset D^2 && 'auf' \\ &\rho_2 = \{(b, a), (c, a), (c, b), (e, d)\} \subset D^2 && 'unter' \\ &\rho_3 = \{a, d\} \subset D && 'frei' \end{aligned} \tag{3.2}$$

Die Definitionen aus (3.2) stellen die explizite Spezifikation einer Konzeptualisierung für das in der Abbildung 3.1 gegebene materiale Szenario dar und lässt sich nach Gruber mithin als Ontologie bezeichnen. Die Diskurswelt schließt ohne Berücksichtigung des Tisches die fünf Blöcke a, b, c, d, e ein. Für die Konzeptualisierung aus (3.1) sind nur drei Relationen relevant (ρ_1, ρ_2, ρ_3), deren Bedeutung sich natürlichsprachlich ausdrücken lässt. Für beliebige $d_1, d_2 \in D$ gilt:

$$\begin{aligned} (d_1, d_2) \in \rho_1 \quad &\Leftrightarrow \quad \text{'}d_1 \text{ steht genau auf } d_2\text{'} \\ (d_1, d_2) \in \rho_2 \quad &\Leftrightarrow \quad \text{'}d_1 \text{ liegt irgendwo unter } d_2\text{'} \\ (d_1) \in \rho_3 \quad &\Leftrightarrow \quad \text{'}d_1 \text{ steht frei'} \end{aligned} \tag{3.3}$$

Genesereth und Nilsson betonen, dass für einen Gegenstandsbereich unterschiedliche Konzeptualisierungen möglich sind - je nach Sinn und Zweck, der mit der Konzeptualisierung verfolgt wird. So findet z.B. der Tisch oder die Farben der Blöcke in (3.2) keine Berücksichtigung. Die Autoren nennen dies 'ontologische Unverbindlichkeit'.[9]

[9] [Genesereth 1989], S. 19

Wenn man nun den Elementen aus D Konstanten und den Relationen aus R Prädikate zuordnet, lassen sich über (D, R) prädikatenlogische Ausdrücke bilden, mit denen sich dann deklaratives Wissen über den Gegenstandsbereich formalisieren lässt. Beispiele hierfür sind die folgenden axiomatischen Ausdrücke mit einem Prädikatenkalkül 1. Ordnung:

$$\begin{gathered} \bigwedge_{d_1,d_2 \in D} auf(d_1, d_2) \Rightarrow unter(d_2, d_1) \\ \bigwedge_{d_1,d_2,d_3 \in D} unter(d_1, d_2) \wedge unter(d_2, d_3) \Rightarrow unter(d_1, d_3) \quad \text{(transitiv)} \end{gathered} \tag{3.4}$$

Guarino und Giarette kritisieren in [Guarino 1995b] den extensionalen Charakter einer Konzeptualisierung gemäß (3.1): Die Relationen werden, wie im Beispiel in (3.2) gezeigt, explizit durch Nennung der Tupel genannt, die für den einen speziellen Fall in Abbildung (3.1) gelten. Wenn die Blöcke in Abbildung (3.1) umgeschichtet würden, müssten die Relationen in (3.2) neu definiert werden und damit die ganze Konzeptualisierung. Die mit den Relationen verknüpften Bedeutungen und Prädikate ('*auf*', '*unter*', '*frei*') wären aber dieselben. Um die fallübergreifende, intensionale Bedeutung einer Konzeptualisierung zu erhalten, wird in [Guarino 1998] vorgeschlagen, die Menge W der möglichen Fälle w für einen Gegenstandsbereich mit in Betracht zu ziehen. W heißt in [Guarino 1998] Welt ('world') und ist vergleichbar mit dem Zustandsraum aus der Systemtheorie. Guarino und Giarette schlagen deshalb die Einführung der konzeptualen Relation vor:

- Die folgende Abbildung

$$\underline{\rho} : W \mapsto \bigcup_{n \in \mathbb{N}} \mathsf{P}(D^n) \tag{3.5}$$

  heißt konzeptuale Relation.

Mit der konzeptualen Relation kann die Tatsache formalisiert werden, dass das Verständnis von einem bestimmten Gegenstandsbereich auch für verschiedene Fälle und Situationen Geltung besitzt. Unter Einbeziehung dieser Relation kommt man dann zu einer erweiterten Konzeptualisierungsdefinition:

- Das 3-Tupel $\mathcal{C}$ mit

$$\mathcal{C} = (D, W, \mathcal{R}) \tag{3.6}$$

  heißt Konzeptualisierung. D ist eine Menge von Objekten eines Gegenstandsbereiches und heißt Diskurswelt. W ist die Menge aller möglichen Fälle w für einen Gegenstandbereich und heißt Welt. $\mathcal{R}$ ist eine Menge konzeptualer Relationen $\underline{\rho}$.

Mit Gleichung (3.1) und (3.6) liegen nun zwei konkurrierende Definitionen für den Begriff *Konzeptualisierung* vor, die folgendermaßen zusammenhängen:

$$K_{w,\mathcal{C}} = (D, R_{w,\mathcal{C}}) \quad R_{w,\mathcal{C}} = \{\underline{\rho}(w) \mid \underline{\rho} \in \mathcal{R} \wedge w \in W\} \quad \mathcal{C} = (D, W, \mathcal{R}) \tag{3.7}$$

Dies bedeutet also, dass eine Konzeptualisierung $\mathcal{C}$ nach (3.6) für einen Fall $w \in W$ in eine Konzeptualisierung $K_{w,\mathcal{C}}$ nach (3.1) übergeht. Nach [Guarino 1998] wird $K_{w,\mathcal{C}}$ auch als '*world structure*' bezeichnet, also als (algebraische) Struktur für die Welt im Falle w. Innerhalb dieser Arbeit werden deshalb Konzpetualisierungen nach (3.1) als *fallspezifisch* oder *fallabhängig* bezeichnet, um sie damit von Konzeptualisierungen nach (3.6) unterscheiden zu können.

Beispiel

Die Abbildung 3.2 zeigt das aus Abbildung 3.1 bekannte Arrangement von Blöcken. Weiterhin ist dort ein Koordinatensystem angegeben, um die die absolute Position der Blöcke zu spezifizieren. Die Koordinaten ermöglichen die Definition von möglichen Fällen.

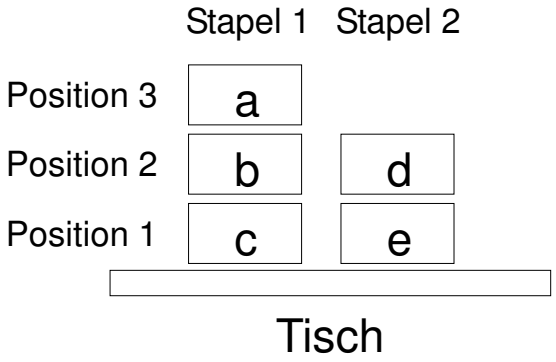


Abbildung 3.2: Erweiterung der Konzeptualisierung

Als Diskurswelt dienen wieder wie in (3.2) die Blöcke.

$$D = \{a, b, c, d, e\} \tag{3.8}$$

Für einen Fall $w \in W$ nehmen die Blöcke $d \in D$ bestimmte Positionen ein. Daraus ergibt sich die Menge möglicher Fälle W:

$$\begin{aligned} W &= \{w_1, \cdots, w_n\} \\ w_1 &= < (a, \overline{31}), (b, \overline{21}), (c, \overline{11}), (d, \overline{22}), (e, \overline{12}) > \\ &\cdots \\ w_n &= < \quad \cdots \quad > \end{aligned} \tag{3.9}$$

Die Relationen, die sich auf das aktuelle Arrangement der Blöcke beziehen, lassen sich nun als konzeptuale Relationen definieren. Hierzu zählen z.B. die Relationen aus Gleichung (3.3):

$$\begin{aligned} \mathcal{R} = \{\underline{\rho}_{auf}, \underline{\rho}_{unter}, \underline{\rho}_{frei}\} \quad \text{mit} \\ \underline{\rho}_{auf} : W \mapsto D^2 \\ \underline{\rho}_{unter} : W \mapsto D^2 \\ \underline{\rho}_{frei} : W \mapsto D \end{aligned} \tag{3.10}$$

So gilt z.B. für den in Abbilung (3.2) gezeigten und in (3.9) beschriebenen Fall w_1:

$$\begin{aligned} \underline{\rho}_{auf}(w_1) &= \{(a,b),(b,c),(d,e)\} \\ \underline{\rho}_{unter}(w_1) &= \{(b,a),(c,b),(c,a),(e,d)\} \\ \underline{\rho}_{frei}(w_1) &= \{(a,d)\} \end{aligned} \tag{3.11}$$

Dies entspricht den Relationen der zu Anfang des Kapitels in (3.2) gezeigten fallspezifischen Konzeptualisierung.

Mit der Diskurswelt D aus (3.8), den Fällen W aus (3.9) und den Relationen $\mathcal{R}$ aus (3.10) ergibt sich die folgende Konzeptualisierung:

$$\mathcal{C} = (D, W, \mathcal{R}) \tag{3.12}$$

Während Gruber sich bei seiner Ontologie-Definition auf die Konzeptualisierung stützt, wird in [Guarino 1998] zusätzlich noch der sprachabhängige Aspekt von Ontologien betrachtet. Die Autoren nennen eine Ontologie ein in einer logischen Sprache L formuliertes System von Axiomen, mit dem eine Konzeptualisierung approximiert werden kann.

Sei L eine logische Sprache mit einer Menge V an Konstanten c und Prädikaten p. Ein sprachliches Modell M ist dann die Interpretation dieser Sprache bezüglich einer fallspezifischen Konzeptualisierung aus (3.1):

- Das 2-Tupel

$$\begin{aligned} M = (K_{w,\mathcal{C}}, I) \quad \text{mit} \\ I : V \mapsto D \cup R_{wC} \quad K_{w,\mathcal{C}} = (D, R_{w,\mathcal{C}}) \end{aligned} \tag{3.13}$$

  heißt sprachliches Modell für die (fallspezifische) Konzeptualisierung K_{wC}.

Gemäß dieser Definition stellt (3.2) ein sprachliches Modell dar. Es werden die Prädikaten '*auf*', '*unter*' und '*frei*' (am rechten Rand der Gleichung notiert) den Relationen ρ_1, ρ_2 und ρ_3 der extensionalen Konzeptualisierung K zugeordnet.

Analog zu dem sprachlichen Modell für eine fallspezifische Konzeptualisierung kann eine logische Sprache und ihr Vokabular V auch bezüglich einer Konzeptualisierung nach (3.6) interpretiert werden:

- Das 2-Tupel

$$\begin{aligned} &\mathcal{M} = (\mathcal{C}, \mathcal{I}) \quad \text{mit} \\ &\mathcal{I} : V \mapsto D \cup \mathcal{R} \quad \mathcal{C} = (D, W, \mathcal{R}) \end{aligned} \tag{3.14}$$

  heißt ontologische Festlegung der Sprache L bzgl. der Konzeptualisierung $\mathcal{C}$.

Ein sprachliches Modell $M = (K_{w,\mathcal{C}}, I)$ ist dann mit der ontologischen Festlegung $\mathcal{M} = (\mathcal{C}, \mathcal{I})$ kompatibel, wenn folgendes gilt:

$$\begin{aligned} &1) \quad K_{w,\mathcal{C}} \in \{(D, R_{w,\mathcal{C}}) \mid w \in W\} \\ &2) \quad \bigwedge_{c \in V} I(c) = \mathcal{I}(c) \\ &3) \quad \bigvee_{w \in W} \bigwedge_{p \in V} \mathcal{I}(p) = \underline{\rho} \wedge \underline{\rho}(w) = I(p) \end{aligned} \tag{3.15}$$

Die Bedingung 1) in (3.15) meint, dass $\mathcal{C} = (D, W, \mathcal{R})$ für einen Fall w in $K_{w,\mathcal{C}}$ übergeht.

In Bedingung 2) wird gefordert, dass I und $\mathcal{I}$ alle Konstanten $c \in V$ auf die gleichen Objekte $d \in D$ abbilden.

In Bedingung 3) wird gefordert, dass es einen Fall $w \in W$ gibt, für den I und $\mathcal{I}$ alle Prädikate $p \in V$ auf dieselben Relationen $\rho \in R_{wC}$ abbilden.

Man kann nun leicht nachvollziehen, dass das sprachliche Modell in (3.2) mit der ontologischen Festlegung $\mathcal{M}$, die in einem solch einfachen Beispiel wie in Abbildung (3.1) ja durch eine natürlichsprachliche Definition (3.3) informal festgelegt werden kann, kompatibel ist. Wie im folgenden gezeigt wird besteht ein großes Problem nun darin, dass sprachliche Modelle eine Konzeptualisierung bzw. deren ontologische Festlegung im Allgemeinen nur approximieren können.

Die Menge aller mit der ontologischen Festlegung $\mathcal{M}$ kompatiblen Modelle M kann dann zur Menge $I_{\mathcal{M}}(L)$ zusammengefaßt werden:

- Die Menge

$$I_{\mathcal{M}}(L) = \{M \mid M \text{ kompatibel zu } \mathcal{M}\} \tag{3.16}$$

  heißt Menge der intendierten Modelle von L bezüglich $\mathcal{C}$.

Nach [Guarino 1998] ist eine Ontologie dann ein System von logischen Aussagen in einer Sprache L, mit dem die intendierten Modelle $I_{\mathcal{M}}(L)$ bezüglich der ontologischen Festlegung $\mathcal{M}$ approximiert werden.

- Das in einer Sprache L formulierte System von logischen Aussagen (Axiomen) $\mathcal{O}_L$ mit

$$\mathcal{O}_L \underset{approx}{\rightsquigarrow} I_{\mathcal{M}}(L) \tag{3.17}$$

heißt Ontologie für die Sprache L.

Im Allgemeinen kann mit einer Ontologie weder die Konzeptualisierung $\mathcal{C}$ noch die Menge der intendierten Modelle $I_{\mathcal{M}}(L)$ genau spezifiziert werden. Vielmehr entspricht eine Ontologie $\mathcal{O}_L$ einer Menge sprachlicher Modelle $M_i = (K_i, L)$, die die intendierten Modelle $I_{\mathcal{M}}(L)$ enthält:

$$\mathcal{O}_L = \{M_1, \cdots, M_n\} \supseteq I_{\mathcal{M}}(L) \tag{3.18}$$

Im besten Fall ist $\mathcal{O}_L = I_{\mathcal{M}}(L)$. Eine genaue Spezifikation der intendierten Modelle ist aber in den meisten Fällen schwer oder mit sehr großem Aufwand möglich oder gar nicht erwünscht, z.B. weil eine Ontologie abstrahiert oder Zusammenhänge vereinfachen soll. In der Praxis wird man versuchen, mit einer Ontologie absurde oder sinnwidrige Fälle auszuschliessen (z.B. dass ein Block nie auf sich selbst stehen kann) und damit $\mathcal{O}_L$ in Richtung $I_{\mathcal{M}}(I)$ einzugrenzen.

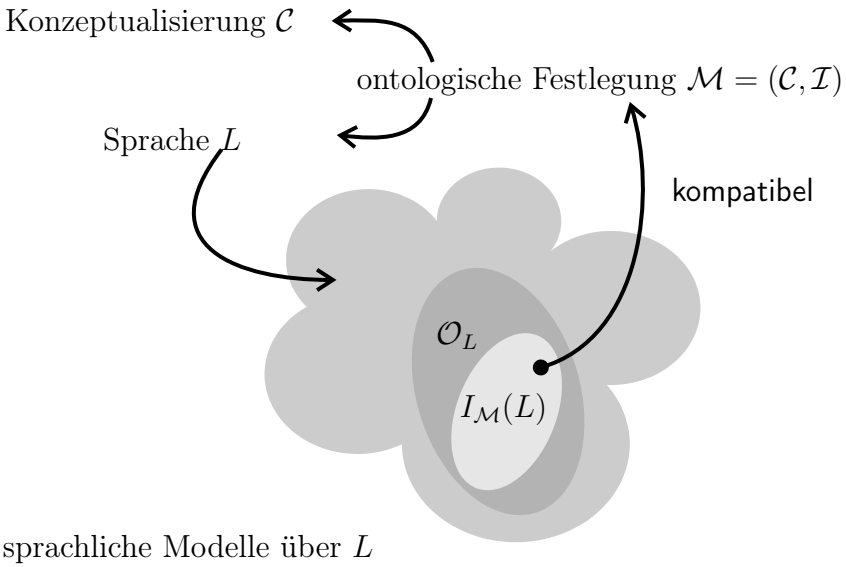


Abbildung 3.3: Ontologien approximieren intendierte Modelle (nach [Guarino 1998])

Dieser Zusammenhang ist in der Abbildung 3.3 grafisch dargestellt. Die Skizze zeigt die ontologische Festlegung einer Sprache L bezüglich der Konzeptualisierung $\mathcal{C}$, siehe auch

(3.14). Weiterhin sind die intendierten Modelle $I_{\mathcal{M}}(L)$ gezeigt, die mit dieser Festlegung kompatibel sind. Eine Ontologie $\mathcal{O}_L$ approximiert diese intendierte Modelle mehr oder weniger gut.

Das folgende Beispiel zeigt einen Ausschnitt aus einer möglichen Ontologie für den Gegenstandsbereich, der mit den auf einem Tisch gestapelten Blöcken befasst ist. Es wird hier als Sprache CycL benutzt, die in Kapitel 3.1.4 noch kurz erläutert wird. Mit dieser Sprache werden logische Ausdrücke in Präfix-Notation erstellt.

Beispiel

```
...
(#$implies
     (#$and(
          (#$isa ?LOWER #$Block)
          (#$auf ?UPPER ?LOWER)
          )
     )
     (#$isa ?UPPER #$Block)
)


(#$implies
     (#$isa ?UPPER #$Block)
     (#$thereExistAtMost 1 ?LOWER
          (#$auf ?UPPER ?LOWER)
     )
)
...
```

Die erste Implikation drückt aus, dass auf Blöcken nur andere Blöcke stehen können (also keine Tische oder sonstiges). Die zweite Implikation meint, dass Blöcke immer nur genau auf einem Element stehen können. Welcher Art dieses zweite Element ist, wird in dieser Implikation nicht festgehalten.

Dieser Teil einer Ontologie schränkt die möglichen Welten von Blöcken bereits ein, lässt aber trotzdem noch Fälle (d.h. sprachliche Modell) zu, die absurd sind - so zum Beispiel, dass ein Block auf sich selbst steht. Für eine spezielle Andwendung müsste man nun entscheiden, ob die Ontologie die intendierten sprachlichen Modelle genau genug approximiert oder ob sie weiter eingeschränkt werden muss.

3.1.2 Ontologische Konzepte

Trotz der unterschiedlichen Auffassungen und formalen Definitionen des Begriffs Ontologie, kann man einige Konzepte als typisch bzw. notwendig für Ontologien bezeichnen. Sie bilden die Grundlage für die Wissensmodellierung und -repräsentation mit Ontologien.

3.1.2.1 Konzeptualisierung

Die Konzeptualisierung ist unabhängig von ihrer extensionalen oder intensionalen Bedeutung (siehe Gleichungen (3.1), (3.6)) ein grundlegendes Konzept von Ontologien. Die Konzeptualisierung besteht in der Einteilung eines Gegenstandsbereichs in eine Diskurswelt und der anschließenden Definition von Relationen über der Diskurswelt. Dieses Vorgehen findet sich auch in anderen Wissensmodellierungsverfahren und kann daher als allgemeingültiges Prinzip der Wissensmodellierung bezeichnet werden. Owsnicki-Klewe, v. Luck und Nebel zeigen z.B. in [Görz 2003], wie Wissen mit mitteln der Prädikatenlogik modelliert werden kann und folgen hierbei dem Konzeptionalisierungsprinzip.

Neben der Konzeptualisierung gibt es weitere Konzepte, um die eine Reihe unterschiedlicher Ontologie-Ansätze erweitert wurden. Als wichtigstes sei an dieser Stelle die Taxonomisierung genannt. Konzepte wie Vererbung, Instanzierung (Schema-Ausprägung) und Metaisierung lassen sich daraus ableiten, wie noch gezeigt wird.

3.1.2.2 Taxonomien

Ein grundlegendes Prinzip der Wissensstrukturierung besteht darin, übergeordnete Seinskategorieen in die Diskurswelt D aufzunehmen und diese durch spezielle binäre Relationen $\rho_{ist-ein} \subset D^2$ zu verknüpfen, die eine 'ist-ein' ('is-a') Bedeutung implizieren. Es entstehen somit Seinshierarchien, die auch Taxonomien genannt werden. Bei den Kategorien einer Taxonomie handelt es sich i.A. um abstrakte, inhaltlich bestimmte Begriffe, die auch intensional genannt werden können. Die intensionalen Elemente einer Konzeptualisierung stehen im Gegensatz zu den extensionalen, die gegenständlich bestimmt sind und den realen Umfang eines intensionalen Begriffs bezeichnen. Ein Beispiel für ein intensionales/extensionales Begriffspaar wäre z.B. Farbe/blau. Die einer Taxonomie unterworfene Diskurswelt lässt sich also in intensionale und extensionale Elemente einteilen[10], wobei auch diese Einteilung kontextbezogen und nicht allgemeingültig ist.

$$D' = \{D_{int}, D_{ext}\} \tag{3.19}$$

Beispielsweise könnte die Diskurswelt des Szenarios aus Abbildung (3.1) um intensionale Elemente wie z.B. 'Bauklotz', 'Kugel', 'Pyramide', 'Einrichtungsgegenstand' usw. erweitert werden. Entsprechende extensionale Elemente der Diskurswelt sind dann die bekannten Blöcke $a, b, \cdots, e$ und als spezieller Einrichtungsgegenstand, auf dem die Klötze

[10] In der Objektorientierung heißen die intensionalen Elemente Klassen und die extensionalen Elemente Instanzen oder einfach nur Objekte

liegen können, der 'Tisch'. Aufbauend auf diesen Begriffen kann man nun eine Taxonomie für diese Diskurswelt entwerfen. Die 'ist-ein' Relation einer Taxonomisierung besteht im Allgemeinen nur zwischen intensionalen Objekten oder zwischen extensionalen und intensionalen Objekten einer Diskurswelt D':

$$\rho_{ist-ein} \subset D_{int}^2 \bigcup D_{ext} \times D_{int} \tag{3.20}$$

Wie später noch gezeigt wird, entspricht der erste Fall der Vererbung, der zweite der Instanzierung.

Weiterhin ist die Relation $\rho_{ist-ein}$ bei vielen Ontologien nicht-konzeptual, d.h. unabhängig von einem gegebenen Fall. In unserem Beispiel aus Abbildung (3.1) würde dies bedeuten: Ein *Block* ist ein *Block*, egal wo er steht. Einige Beschreibungsmittel für Ontologien hingegen erlauben die Modellierung konzeptualer 'ist-ein' Relationen, so z.B. die Datenmodellierungssprache EXPRESS aus den STEP-Normen zur Produktdatenmodellierung.

$$\underline{\rho}_{ist-ein} : W \mapsto D_{int}^2 \bigcup D_{ext} \times D_{int} \tag{3.21}$$

Gleichung (3.21) bedeutet also, dass ein Objekt $d \in D'$ zur Laufzeit seine Taxonomiezugehörigkeit ändern kann. Hafner erläutert diese Problematik im Zusammenhang mit Ontologien für molekularbiologische Prozesse. Aufgrund chemischer Prozesse können Reagenzien ihre Kategorie innerhalb einer Taxonomie ändern. Eine eindeutige Produktverfolgung erfordert dann Ontologien, die Kategoriekonversion unterstützen ([Hafner 2000]).

Die in Taxonomien ausgedrückte Zugehörigkeit zu einer höheren Seinskategorie wird in der KI-Literatur im Gegensatz zu dem hier verfolgten Ansatz der binären Relation (vgl. (3.20)) überlicherweise mit unären Relationen $\rho \subset D$ ausgedrückt. Um z.B. auszudrücken, dass es es bei dem Objekt a um einen *Block* handelt, würde man eine Relation $\rho_{Block} = \{a\}$ definieren und ihr das Prädikat *Block* zuordnen:

$$\rho_{Block} = \{a\} \Leftrightarrow Block(a) \tag{3.22}$$

Beide Ansätze sind äquivalent. Jedoch lassen sich Vererbung und Instanzierung mit binären Relationen einfacher veranschaulichen, wie im folgenden gezeigt wird.

3.1.2.3 Vererbung, Instanzierung

Eng mit den Taxonomien hängen die Konzepte *Vererbung* und *Instanzierung* zusammen. Vererbung und Instanzierung sind zunächst einmal gemäß (3.20) spezielle Taxonomie-Relationen, wobei deren Unterscheidung auf der Einteilung der Diskurswelt in intensionale und extensionale Objekte nach (3.19) beruht: Vererbungsrelationen sind taxonomische Relationen zwischen intensionalen Elementen, Instanzierungsrelationen sind taxonomische Relationen zwischen intensionalen und extensionalen Elementen:

$$\begin{aligned} &(d_1, d_2) \in \rho_{ist-ein} \quad \wedge \quad d_1, d_2 \in D_{int} && Vererbung \\ &(d_1, d_2) \in \rho_{ist-ein} \quad \wedge \quad d_1 \in D_{ext}, d_2 \in D_{int} && Instanzierung \end{aligned} \tag{3.23}$$

Bei Gleichung (3.23) handelt es sich natürlich um keine formale Definition der Begriff Vererbung und Instanzierung, sondern um eine notwendige Bedingung. Entscheidend ist, dass beide Relationen Einfluss auf die Eigenschaftsrelationen bzw. Attribute einer Konzeptualisierung besitzen.

Bei der Vererbung wird gewährleistet, dass die Eigenschaften der Elternelemente auf die Kindelemente übertragen werden. Jedes Kindelement ist also Teil aller Eigenschaftsrelationen, an denen auch das Elternelement teilnimmt:

$$(d_1, d_2) \in \rho_{ist-ein} \quad \text{mit} \quad d_1, d_2 \in D_{int}$$
$$\Rightarrow \left((d_2, \bullet) \in \rho_{Attrib,i} \Rightarrow (d_1, \bullet) \in \rho_{Attrib,i} \right) \tag{3.24}$$

Ähnliches gilt für die Instanzierung. Hier wird gewährleistet, dass die Eigenschaften eines Elternelementes bei deren Instanzen ausgeprägt sind

$$(d_1, d_2) \in \rho_{ist-ein} \wedge (d_2, d_2') \in \rho_{Attrib,i} \quad \text{mit} \quad d_2, d_2' \in D_{int}, d_1 \in D_{ext}$$
$$\Rightarrow \bigvee_{d_1' \in D_{ext}} \left((d_1, d_1') \in \rho_{Attrib,i} \wedge (d_1', d_2') \in \rho_{ist-ein} \right) \tag{3.25}$$

Dies lässt sich anschaulich in einem Relationengraphen illustrieren. Abbildung (3.4) zeigt das Elternelement d_2 und seine Instanzierung d_1. Da d_2 eine Eigenschaft d_2' besitzt, muss auch d_1 eine solche Eigenschaft d_1' besitzen, wobei d_1' Instanz von d_2' sein muss.

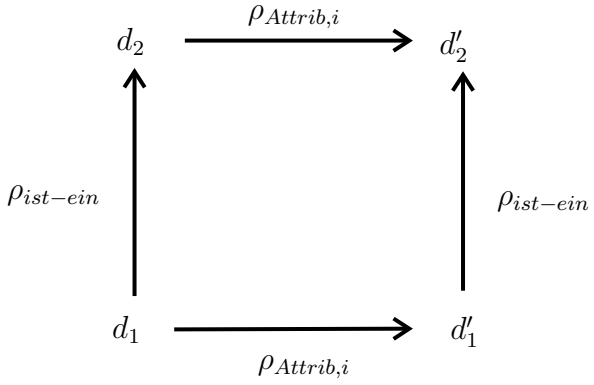


Abbildung 3.4: Relationengraph zur Instanzierung

Bei Instanzierung spricht man auch vom *Schema-Ausprägungs-Mechanismus* [Ortner 2000a]. Das intensionale Element einer Instanzierungsrelation heißt dann Schema, die Instanz als

extensionales Element heißt Ausprägung. Diese Begriffsbildung mit Schema und Ausprägung deutet auf den in Abbildung 3.4 illustrierten Zusammenhang hin, dass das Elternelement d_2 mit seinen Attributen ein Schema vorgibt, das durch die Instanz d_1 ausgeprägt wird.

3.1.3 Integration von Ontologien

Eines der grundlegende Ziele der Informationsmodellierung ist die eindeutige, zweifelsfreie Spezifikation von Anwendungswissen, um damit den Informationsaustausch zwischen Rechneranwendungen zu ermöglichen. Dieser Informationsaustausch ist auch ein vorrangiges Ziel der Forschung auf dem Gebiet der Ontologien. Gruber formuliert dies so:

> *Recent work in Artificial Intelligence (AI) is exploring the use of formal ontologies as a way of specifying content-specific agreements for the sharing and reuse of knowledge among software entities.* [11]

Im Allgeimenen liegen den Anwendungslogiken unterschiedlicher Softwaresysteme auch unterschiedliche Ontologien zugrunde. Die Frage nach dem Informationsaustausch zwischen den Anwendungen lässt sich dann als Integrationsproblem von Ontologien auffassen. Wie in Abschnitt 3.1.1 erwähnt kann eine Ontologie als ein in einer logischen Sprache formuliertes Aussagensystem angesehen werden, mit dem die intendierten Modelle einer Konzeptualisierung approximiert werden können, Gleichungen (3.16) und (3.17).

Zunächst soll der Fall zweier ontologischer Festlegungen $\mathcal{M}_1$ und $\mathcal{M}_2$ bezüglich derselben Sprache L betrachtet werden.

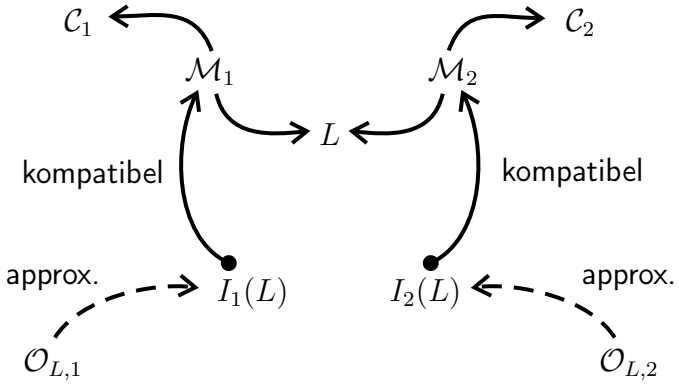


Abbildung 3.5: Integration von Ontologien derselben Sprache L

In [Guarino 1998] wird darauf hingewiesen, dass die zwei Ontologien $\mathcal{O}_{L,1}$ und $\mathcal{O}_{L,2}$ nicht miteinander verträglich sind (i.e. die gleiche ontologische Festlegung besitzen),

[11] [Gruber 1995], S. 907

sobald sie in derselben Sprache formuliert sind[12]. Formal ist die Integration dieser Ontologien nur dann möglich, wenn sich ihre intendierten Modelle überlappen:

$$\begin{aligned} \mathcal{O}_{L,1} \underset{approx}{\rightsquigarrow} I_1(L) \quad \text{und} \quad \mathcal{O}_{L,2} \underset{approx}{\rightsquigarrow} I_2(L) \\ I_1(L) \cap I_2(L) \neq \emptyset \end{aligned} \tag{3.26}$$

Nur diejenigen sprachlichen Modelle, die in die Schnittmenge $I_1(L) \cap I_2(L)$ fallen, können zwischen den unterschiedlichen Anwendungen ausgetauscht werden. Diese Schnittmenge ist im Allgemeinen a priori nicht bekannt. Daher muss der Datenaustausch zwischen den beteiligten Anwendungen von Fall zu Fall auf Konsistenz überprüft werden (siehe ProSTEP Benchmarks).

3.1.3.1 Metaisierung

Um eine konsistente Integration unterschiedlicher Ontologien zu ermöglichen, schlägt Guarino vor, Ontologien von unterschiedlichem Abstraktionsniveau zu entwerfen, die sich aufeinander beziehen, wobei jede Ontologie die Diskurswelten der nächsthöheren Abstraktionsstufe verwenden bzw. verfeinern soll. Die Ontologien lassen sich dann nach ihrem Abstraktionsniveau folgendermaßen einteilen:

- *Top-level Ontologien* mit domänenunabhängiger Konzeptualisierung
- *domänen- oder aufgabenspezifische Ontologien*
- *Anwendungsontologien* mit domänen- und aufgabenspezifischer Konzeptualisierung

Für die Verwendung der Diskurswelten aus Ontologien höherer Abstraktionsstufe kommen zunächst taxonomische 'ist-ein'-Relationen wie Vererbung oder Instanzierung in Frage. Im Gegensatz dazu sind Referenzen auf bereits definierte Diskurswelten nur zwischen Ontologien gleichen Abstraktionsniveaus möglich. Das Verfahren, Objekte zweier Ontologien unterschiedlichen Abstraktionsniveaus durch taxonomische Instanzen-Relationen zu verknüpfen wird auch Metaisierung oder Metamodellierung genannt. Hierunter versteht man i. A. die Definition von 'ist-ein' Relationen zwischen den Objekten zweier Ontologien, die beide die Diskurswelten D und D' besitzen. Ein Objekte $d \in D$ ist dann Instanz eines Objektes $d' \in D'$.

Metaisierung stellt ein mächtiges Konzept zur Wissensstrukturierung dar. Hierauf wird im Kapitel 3.2.2.3 näher eingegangen.

[12] Anschaulich gesprochen könnte dies bedeuten, dass zwei Personen über Bauklötze und deren Anordnung auf einem Tisch sprechen, aber unterschiedliche Dinge damit meinen

3.1.3.2 Strukturverträglichkeit

Neben dem in (3.26) vorgestellten Kriterium zur Ontologieintegration wird in dieser Arbeit ein weiteres Integrationskriterium vorgestellt, das der in dieser Arbeit vorgestellten Methode zur Modellintegration zugrunde liegt. Dazu wird der allgemeine Fall zweier ontologischer Festlegungen $\mathcal{M}_1$ und $\mathcal{M}_2$ untersucht, die sich sowohl in Konzeptualisierung als auch in der Sprache unterscheiden. Dies wird in der Abbildung 3.6 verdeutlicht.

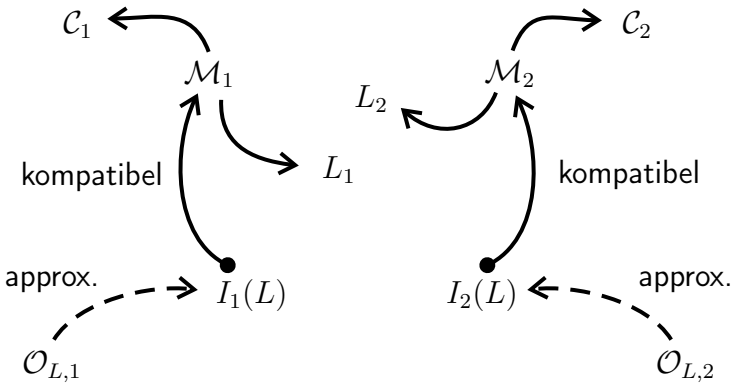


Abbildung 3.6: Integration von Ontologien unterschiedlicher Sprachen und Konzeptualisierungen

Obwohl die Ontologien $\mathcal{O}_{L,1}$ und $\mathcal{O}_{L,2}$ weder in Bezug auf Sprache noch auf Konzept Gemeinsamkeiten besitzen, ist eine Integration durchaus möglich und sinnvoll. Diese Integration beruht auf gewissen strukturellen Ähnlichkeiten, die zwischen den Ontologien herrschen.

Die algebraische Grundlage zur Beschreibung struktureller Ähnlichkeiten oder Symmetrien bildet die Gruppentheorie. Die Gruppentheorie ist ein Teilgebiet der Algebra und befasst sich mit der Beschreibung von algebraischen Stukturen und deren Eigenschaften. Eine wichtige Beziehung, die zwischen algebraischen Strukturen bestehen kann, ist deren Strukturverträglichkeit. Eine Abbildung, die zueinander strukturverträgliche algebraische Strukturen aufeinander abbildet, heisst Homomorphismus (bzw. als bijektive Abbildung Isomorphismus). Die Beschreibung natürlicher Symmetrien auf der Basis von Homo/Isomorphismen ermöglichen eine breite Anwendung der Gruppentheorie auf dem Gebiet der modernen Kern- und Quantenphysik ([Bronstein 1995a]). Die Strukturverträglichkeit von Ontologien wird an dieser Stelle nicht mit den Mitteln der Gruppentheorie formuliert bzw. untersucht. Dies würde erfordern, den Begriff der Konzeptualisierung so zu formulieren, dass er Gruppeneigenschaften aufweist (z.B. innere Verknüpfungen). Dieser Ansatz wird hier nicht verfolgt, wäre aber prinzipiell möglich.

Zwei Ontologien heißen nun strukturverträglich, wenn es relationentreue (strukturverträgliche, homo/isomorphe) Abbildungen zwischen den Konzeptualisierungen der intendierten sprachliche Modelle, die durch die Ontologie approximiert werden, gibt. Mit dem Begriff 'strukturverträglich' werden im Gegensatz zur Gruppentheorie nicht nur

die Abbildungen, die zwischen den Ontologien vermitteln, sondern auch die Ontologien selbst bezeichnet. Damit erleichtert sich die Erläuterung mancher Zusammenhänge.

Die Definition strukturverträglicher Ontologien lässt sich nun folgendermaßen formalisieren. Gemäß Abbildung 3.6 werden zwei Ontologien $\mathcal{O}_{L,1}$ und $\mathcal{O}_{L,2}$ betrachtet, die die intendierten Modelle $I_1(L)$ und $I_2(L)$ approximieren. Nach (3.16) sind dies Mengen von sprachlichen Modellen M_a und M_b, die mit den ontologischen Festlegungen $\mathcal{M}_1$ und $\mathcal{M}_2$ kompatibel sind:

$$M_a \in I_1(L) \qquad M_b \in I_2(L)$$

Da alle sprachlichen Modelle in $I_1(L)$ und $I_2(L)$ nach (3.16) mit $\mathcal{M}_1$ und $\mathcal{M}_2$ kompatibel sind, handelt es sich bei den sprachlichen Modellen M_a und M_b um fallspezifische Ausprägungen der Konzeptualisierungen $\mathcal{C}_1$ und $\mathcal{C}_2$:

$$\begin{aligned} M_a &= (K_{w,\mathcal{C}_1}, I_1) \quad \text{mit} \\ &K_{w,\mathcal{C}_1} = (D_1, R_{w,\mathcal{C}_1}) \\ &R_{w,\mathcal{C}_1} = \{\underline{\rho}(w) \mid \underline{\rho} \in \mathcal{R}_1 \wedge w \in W\} \\ &\mathcal{C}_1 = (D_1, W, \mathcal{R}_1) \end{aligned} \tag{3.27}$$

Analog zu (3.27) gilt für ein sprachliches Modell M_b derselbe Zusammenhang:

$$\begin{aligned} M_b &= (K_{w,\mathcal{C}_2}, I_2) \quad \text{mit} \\ &K_{w,\mathcal{C}_2} = (D_2, R_{w,\mathcal{C}_2}) \\ &\dots \end{aligned} \tag{3.28}$$

Betrachtet werden nun zwei Relationen $\rho_a \in R_{w,\mathcal{C}_1}$ und $\rho_b \in R_{w,\mathcal{C}_2}$. Beide werden als n-stellig angenommen.

$$\rho_a \subset D_1^n \quad \rho_b \subset D_2^n \tag{3.29}$$

Wenn man nun eine nicht notwendigerweise bijektive Abbildung h zwischen den Diskurswelten D_1 und D_2 finden, die dafür sorgt, dass alle Bilder der Tupel von ρ_a in ρ_b enthalten sind, also $\rho_b = h(\rho_a)$ ist, dann heisst h strukturverträglich.

- Sei h eine Abbildung mit $h : D_1 \mapsto D_2$. Weiterhin gilt:

$$\begin{aligned} \bigwedge_{d_{a,1},\cdots,d_{a,n} \in D_1} \Big(&(d_{a,1}, \cdots, d_{a,n}) \in \rho_a \Rightarrow \\ &h(d_{a,1}, \cdots, d_{a,n}) = (h(d_{a,1}), \cdots, h(d_{a,n})) = (d_{b,1}, \cdots, d_{b,n}) \in \rho_b \Big) \end{aligned} \tag{3.30}$$

Die Abbildung h heisst strukturverträgliche Abbildung oder Modelltransformation zwischen den Modellen M_a und M_b.

Das Kriterium der Strukurverträglichkeit in (3.30) bezieht sich auf zwei beliebige Relationen ρ_a und ρ_b in zwei beliebigen Modellen M_a und M_b. Man kann deshalb versuchen, dieses Kriterium in zweierlei Richtung zu verallgemeinern.

1. Zuerst kann gefordert werden, dass (3.30) für alle Relationen $\rho_a \in R_{w,\mathcal{C}_1}$ gelten soll. Man muss also für jede Relation ρ_a ein Äquivalent in $\rho_b \in R_{w,\mathcal{C}_2}$ finden.

2. Eine weitere Forderungen kann darin bestehen, dass (3.30) für alle sprachlichen Modelle $M_{a,i} \in I_1(L)$ gelten soll. Nach (3.15) ist jedes sprachliche Modell $M_{a,i}$ ja eine fallspezifische Ausprägung der ontologischen Festlegung $\mathcal{M}_1$. Somit ergibt sich diese Forderungen, wenn h für alle $w \in W$ gilt.

Die Verallgemeinerung in Fall 1) ist theoretisch möglich, stößt in der Praxis aber auf Schwierigkeiten. Um die Strukturverträglichkeit einer Abbildung zwischen sprachlichen Modellen nachzuweisen, müssten dazu sämtliche Relationen untersucht werden. Da solche Abbildungen in dieser Arbeit tatsächlich untersucht werden sollen, beschränken wir uns hier auf die Strukturverträglichkeit bezüglich ausgewählter Relationen, um so auch praktisch relevante Ergebnisse erarbeiten zu können.

Die Verallgemeinerung in Richtung 2) hat ebenfalls den Nachteil, dass ein eventueller Nachweis der Strukturverträglichkeit für eine Abbildung aufwendig werden kann. Der Vorteil liegt aber darin, dass eine solche strukturverträgliche Abbildung dann zwischen allen intendierten Modellen zweier Ontologien vermittelt. Man kann dann zwei Ontologien strukturverträglich nennen, wenn es eine solche strukturverträgliche Abbildung zwischen diesen Ontologien gibt. Da in dieser Arbeit solche Ontologien untersucht werden sollen, ist diese Verallgemeinerung sinnvoll. Der streng mathematische Nachweis der Strukturverträglichkeit zweier Ontologien wird in dieser Arbeit nicht geführt. Dies würde den Rahmen der Arbeit sprengen. Vielmehr wird die Strukturverträglichkeit durch Anschauung und an Beispielen erläutert.

Zustätzlich zur Verallgemeinerung um alle Fälle $w \in W$ ist es sinnvoll und von praktischer Relevanz, die Implikation in (3.30) in eine Äquivalenz zu überführen. Die Modelltransformation durch h wird damit umkehrbar bzw. linkseindeutig oder auch injektiv (h sei nicht surjektiv, was in der Praxis durchaus der Fall sein kann). Mit der Ausweitung von (3.30) auf beliebige Relationen und auf alle Fälle w plus Äquivalenz ergibt sich folgende Definition.

- Seien $\mathcal{O}_{L,1}$ und $\mathcal{O}_{L,2}$ zwei Ongologien mit

$$\mathcal{O}_{L,1} \underset{\text{approx}}{\rightsquigarrow} I_1(L) \quad \text{und} \quad \mathcal{O}_{L,2} \underset{\text{approx}}{\rightsquigarrow} I_2(L)$$

$$I_1(L) = \{M_i \mid M_i \text{ kompatibel zu } \mathcal{M}_1\} \quad \text{und} \quad I_2(L) = \{M_j \mid M_j \text{ kompatibel zu } \mathcal{M}_2\}$$

$$\mathcal{M}_1 = (\mathcal{C}_1, \mathcal{I}_1) \quad \text{und} \quad \mathcal{M}_2 = (\mathcal{C}_2, \mathcal{I}_2)$$

$$\mathcal{C}_1 = (D_1, W, \mathcal{R}_1) \quad \text{und} \quad \mathcal{C}_2 = (D_2, W, \mathcal{R}_2)$$

  Weiterhin gebe es eine injektive Abbildung $h : D_1 \mapsto D_2$, die bezüglich einer Reihe konzeptualer Relationen $\underline{\rho}_{a,k} \in \mathcal{R}_1$ und $\underline{\rho}_{b,l} \in \mathcal{R}_2$ strukturverträglich ist, so dass gilt:

$$\bigwedge_{w \in W} \bigwedge_{d_{a,1}, \cdots, d_{a,n} \in D_1} \Big((d_{a,1}, \cdots, d_{a,n}) \in \underline{\rho}_{a,k}(w) \Leftrightarrow h(d_{a,1}, \cdots, d_{a,n}) \in \underline{\rho}_{b,l} \Big) \tag{3.31}$$

  Die Ontologien $\mathcal{O}_{L,1}$ und $\mathcal{O}_{L,2}$ heissen dann strukturverträglich. Die Abbildung h ist eine strukturverträgliche Modelltransformation.

Die Strukturverträglichkeit von Ontologien sei an einem einfachen Beispiel verdeutlicht.

Beispiel

Gegeben sei eine Ontologie, die die sprachlichen Modelle für die möglichen Konstellationen fünf verschiedener Blöcke approximiert. Ein mögliches Blockarrangement zeigt die bereits bekannte Abbildung 3.1. Die Sprache, auf die sich diese Ontologie bezieht, bestehe aus den Konstanten *a*, ... ,*e* und den Prädikaten *steht auf* und *unter*. Eine weitere Ontologie aus dem Bereich der Biologie approximiere die möglichen sprachlichen Modelle zur Beschreibung eines Ökosystems mit fünf Organismen. Die Sprachkonstanten der zweiten Ontologie für die Organismen sind *A*, ... ,*E*. Die Prädikate lauten *frisst* und *ist Nahrungsgrundlage von*. Beide Ontologien könnten dadurch strukturverträglich aufeinander abgebildet werden, dass die Blöcke *a* bis *e* den Organismen *A* bis *E* zugeordnet werden. Den Relationen *steht auf* und *unter* entsprechen dann die Relationen *frisst* und *ist Nahrungsgrundlage von*. Die folgende Abbildung 3.7 zeigt dann zwei mögliche sprachliche Modelle, die miteinander strukturverträglich sind:

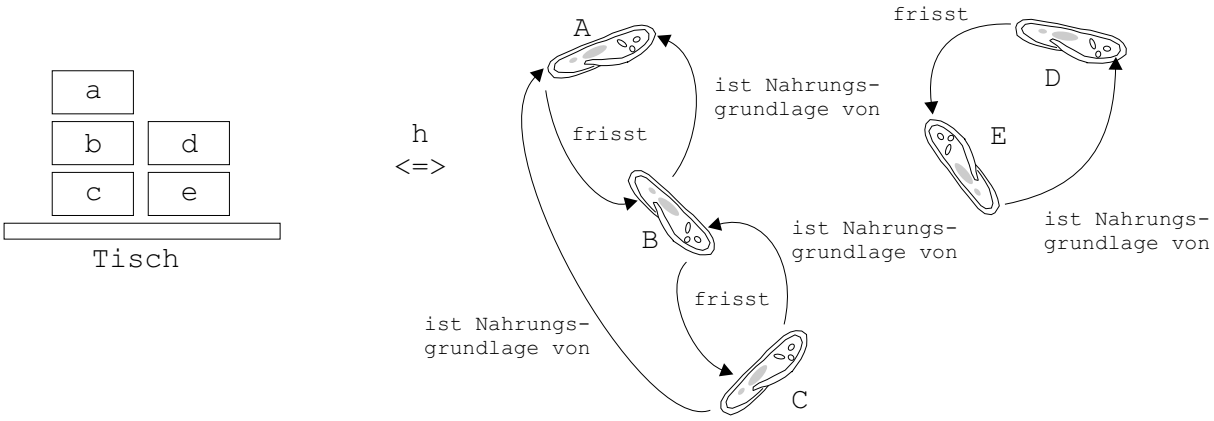


Abbildung 3.7: sprachliche Modelle strukturverträglicher Ontologien

In den späteren Kapiteln dieser Arbeit werden noch einige Ontologien aus dem Bereich der Automatisierungstechnik präsentiert, die strukturverträglich integriert werden können.

Die Vorteile der Integration von Ontologien aufgrund von struktureller Ähnlichkeit hat u.a. folgende Vorteile:

- Die Strukturverträglichkeit von Ontologien ermöglicht eine einfache Transformation und damit Integration sprachlicher Modelle. In vielen Fällen wird diese Transformation nach (3.31) injektiv sein, jedoch in der Praxis nicht immer bijektiv aufgrund fehlender Surjektivität.
- Strukturverträglichkeit kann als Konsistenzkriterium herangezogen werden. Unabhängig voneinander erzeugte Modelle verschiedener Ontologien können anhand dieser Kriteriums auf Konsistenz überprüft werden.
- Strukturvertäglichkeit schafft Klarheit über die Bedeutung einer Ontologie. Durch den Bezug zu bereits bekannten Konzepten und Sprachen wird deren Bedeutung klar. Dies zeigt sich z.B. anhand von Petrinetzmodellen, die als graphische Repräsentationen verschiedener ontologischer Festlegungen dienen kann.
- Die Einhaltung von Struktruverträglichkeitskriterien kann als zielführende Modellierungsmethode genutzt werden. Bestehende strukturelle Basismodelle weisen den Weg bei der Erstellung neuer Modelle. Dies wird im Kapitel 7 als strukturtreues Modellieren näher untersucht.
- Modellverwaltung. Strukturverträgliche Modelle können anhand einer Referenzstruktur partitioniert und verwaltet werden. Derart partitionierte Gesamtmodelle lassen sich z.B. in einer kollaborativen Entwurfsumgebung editieren und weiterverarbeiten ([Däubler 2000a], [Däubler 2000b]).

3.1.4 Stand der Technik

Abschließend werden in diesem Kapitel noch einige Formalismen/Sprachen vorgestellt, die die Spezifikation von Ontologien ermöglichen. Diese Auswahl ist unvollständig und zeigt nur einen Ausschnitt aus dem weiten Spektrum von ontologischen Ansätzen zur Wissensmodellierung.

- KIF (*Knowledge Interchange Format*) - KIF [Genesereth] wurde Anfang der 90'er Jahre an der Stanford University entwickelt. Es dient der Spezifikation von Konzeptualisierungen ganz im Sinne der Definition in (3.1). Es ermöglicht die Definition von Diskurswelten mit Objekten und Klassen, denen die Objekte angehören. Relationen und Funktionen über diesen Diskurswelten können ebenfallse definiert werden. Deren Signatur muss aber von vornherein nicht feststehen. Vererbung wird nicht unterstützt. Weiterhin besteht in KIF die Möglichkeit, deklaratives Wissen zu formulieren. KIF bietet dazu alle üblichen Konstrukte der Prädikatenlogik wie Konstanten, Prädikate, Quantoren, logische Verknüpfungen usw. Die formale Syntax von KIF liegt bereits als Entwurf für einen ANSI-Standard vor.

- CycL - CycL [Cycorp] ist die Grundlage der Cyc-Wissensbasis. Diese von der Firma Cycorp entwickelte Wissensbasis zielt darauf ab, Alltags- und Allgemeinwissen möglichst vollständig und formal zu speichern. Anwendungen für diese Wissensbasis sind z.B. semantische Suchmaschinen für das Internet, die über die reine Stichwortsuche hinaus die inhaltliche Suche ermöglichen. CycL unterstützt die Definition beliebiger Relationen und Funktionen mit fester Signatur. Wohlgeformtheitsregeln ermöglichen es, die Signaturen von Relationen und Funktionen festzulegen bzw. zu überprüfen. Mit den vordefinierten binären `#$isa`-Relationen kann man Taxonomiene aufbauen. Vererbung und Instanzierung gibt es aber nicht. Weiterhin können prädikatenlogische Aussagen formuliert werden.

- OIL (Ontology Inference Layer) - OIL [Horrocks et al.] wird von den Autoren als Standard zum Austausch zur Spezifikation von Ontologien vorgeschlagen. Der Sprachumfang ist zunächst in drei Ebenen unterteilt, die aber in Zukunft erweitert werden sollen: Core OIL, Standard OIL, Instance OIL. Eine OIL-Spezifikation besteht aus einem *ontology container* und einer oder mehreren *ontology definitions*. Der *ontology container* enthält nur administrative Angaben über die spezifizierte Ontologie (z.B. Autor, Sprache, Datum, Urheberrechte) gemäß dem *Dublin Core Metadata Element Set*. Die eigentliche Spezifikation befindet sich im Abschnitt *ontology definitions*. Hier werden die intensionalen Objekte der Diskurswelt, Klassen genannt, und die Taxonomien zwischen diesen Objekten definiert. Instanzierung ist im Sprachumfang von Core OIL nicht möglich. In anderen Ontologien definierte Objekte können importiert werden. Eigenschaftsrelationen werden durch sogenannte *slots* definiert. Diese können wie Klassen auch taxonomisch geordnet werden. Weiterhin können *slot-constraints* definiert werden. Dies sind Regeln, mit denen der mögliche Umfang an Instanzen, die an einer Relation beteiligt sind, ein-

gegrenzt werden kann. In [Klein 2000] wird gezeigt, wie OIL-Spezifikationen für WWW-Anwendungen mit XML implementiert werden können.

- OndoEdit - Hier handelt es sich um einen von der Firma Ontoprise entwickelten Editor zur Spezifikation von Diskurswelten und Relationen [Ontoprise]. Taxonomien mit Vererbung und Instanzierung werden unterstützt. Die intensionalen Objekte einer Diskurswelt heißen Konzepte, die extensionalen Instanzen. Die Angabe administrativer Daten zur Ontologie ist möglich und ähnelt dem *ontology container* von OIL. Axiome können nur in eingeschränktem Umfang spezifiziert werden. Relationen-Axiome ermöglichen es, zu spezifizieren, ob eine gegeben Relation symmetrisch oder transitiv ist, oder ob eine inverse Relation existiert.

Die Ontologie-Thematik gewinnt insbesondere im Zusammenhang mit der Vision vom Semantischen Web an Aktualität. Laut Berners-Lee zeichnet sich das Semantische Web folgendermaßen aus:

> *The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation*[13]

Beispiele

Prominentes Beispiel für die neue, erweiterte Funktionalität des Semantischen Webs sind intelligente Suchfunktionen (siehe oben, CycL). Während in heutigen Browsern Suchfunktionen rein textuell ausgeführt werden, wäre im Semantischen Web die inhaltliche Suche möglich - die Suchbegriffe müssen in der gesuchten Web-Ressource nicht explizit auftauchen [Mintert 2003].

1. Suche: 'gefährliche outdoor-Sportart' -> Ergebnis: Web-Seite über Freeclimber
2. Suche: 'italienische Rotweine' -> Ergebnis: Web-Seite mit Preisliste für Chianti-Weine

---

Stand der Technik ist bisher das wohldefinierte *Format* von Web-Inhalten z.B. in Form der vom WWW-Konsortium standardisierten Technologien wie HTML, XHTML oder

[13] [Berners-Lee 2001]

XML. In Zukunft soll es zusätzlich möglich sein, die *Bedeutung* bzw. den *Informationsgehalt* von Web-Inhalten auf der Basis von Ontologien definieren zu können. Darum bemüht sich das WWW-Konsortium vermehrt um die Standardisierung von Sprachen/Formalismen zur Spezifikation von Ontologien. So existiert bereist eine eigene *Web Ontology Working Group, WebOnt* [W3C 2003c]. Die nachfolgenden Sprachen sind im Umfeld des Semantischen Web entstanden und teilweise bereits WWW-Standard.

- RDF (*Resource Description Framework*) - RDF [W3C 2003b] liegt bereits als WWW-Standard vor und dient der Repräsentation von Informationen über Web-Ressourcen. RDF bietet in seineme Sprachumfang nur ganz elementare Mittel zur Konzeptualisierung (Taxonomien mit Klassen/Vererbung und Instanzierung, binäre Eigenschaften) und kann deshalb als *'light weight ontology system'* bezeichnet werden[14]. Die Formulierung von Einschränkungen und prädikatenlogischer Axiome ist nur eingeschränkt möglich. Die Syntax von RDF ist XML. Alle Konstanten und Prädikate in RDF können können Web-Ressourcen in Form von URI's (Uniform Ressource Identifier) sein. Damit ist die direkte Anwendung von RDF auf web-orientierte Applikationen möglich. Die größte Bedeutung von RDF liegt aber wohl darin, dass RDF die Grundlage von DAML und OWL bildet. Beide Sprachsysteme sind Erweiterung von RDF und beruhen auf dessen Syntax.

- DAML (*DARPA Agent Markup Language*) - DAML [DARPA 2003] ist eine von DARPA (Defense Advanced Research Project Agency) für das Semantische Web entwickelte Ontolgiesprache. DAML wurde um Konstrukte der OIL (siehe oben) erweitert und im Jahre 2001 als DAML+OIL [Feynberg 2001] beim WWW-Konsortium zur Standardisierung eingereicht. DAML+OIL ist zwar kein WWW-Standard geworden, ist aber in die Entwicklung von OWL, dem heuten Web-Ontologiestandard, eingeflossen. DAML beruht auf der RDF-Syntax und erweitert diese um viele Elemente. Diese Erweiterungen betreffen hauptsächlich Axiome prädikatenlogischer Form (Inklusionen, Relationeneigenschaft wie Transitivität und Symmetrie), Modelleinschränkungen durch Kardinalitäten und Mengenoperationen über Klasseninstanzen.

- OWL (*Web Ontology Language*) - OWL [W3C 2003a] ist der jüngste Ontologie-Standard des WWW-Konsortiums. OWL ist eine Erweiterung von RDF und basiert auf DAML-OIL. Letzteres lässt sich auch in OWL überführen [15]. OWL ist unterteilt in drei Sprachuntermengen. *OWL Lite* ist vergleichbar mit dem Sprachumfang von RDF und ermöglicht Taxonomien und einfache Einschränkungen (Kardinalitäten). *OWL DL* steht für *description logic* und ist vergleichbar mit dem Sprachumfang von DAML+OIL. Neben den Taxonomien ist die Spezifikation vieler Relationeneigenschaften und Einschränkungen (Axiome) möglich. Der Sprachumfang von OWL DL garantiert nach [W3C 2003a] die Entscheidbarkeit/Vollstängikeit

[14] [W3C 2003b], 1. Absatz
[15] [W3C 2003c], Relationship with DAML+OIL

(alle Ausdrücke lassen sich in endlicher Zeit auswerten). *OWL Full* stellt den vollen Sprachumfang dar und ermöglicht Metaisierung: Klassen sind gleichzeitig Instanzen anderer Klassen (sogenannte Metaklassen, siehe Abschnitt 3.1.3.1). Für diesen Sprachumfang können dann keine Berechenbarkeitseigenschaften garantiert werden.

3.2 Objektorientierung

Auch die Objektorientierung beinhaltet wie die eben beschriebenen Ontologien unterschiedliche Konzepte, um Anwendungswissen strukturieren und repräsentieren zu können, wobei die in der Objektorientierung genutzten Konzepte eng mit denen der Ontologien zusammenhängen.

Historisch gesehen hat die Objektorientierung ihren Ursprung in der in den Jahren 1962 und 1967 von den norwegischen Wissenschaftlern Ole-Johan Dahl und Kristen Nygaard entwickelten Programmiersprache *SIMULA* (Simulation Language). Wie im Name bereits deutlich wird, war es das Ziel von Dahl und Nygaard, eine Sprache zur Simulation ereignisdiskreter Systeme zu entwickeln. SIMULA wurde aber nicht nur als reine Programmiersprache, sondern insbesondere auch als formales Mittel zur Systembeschreibung in technisch-naturwissenschaftlichen Anwendungen konzipiert und herangezogen. Dies geht aus folgendem Zitat von Kristen Nygaard aus dem Jahre 1963 hervor.

> *... there is a definite need for a precise language which allows a description of a network in terms of standardized and generally accepted concepts. Such a description should force the research worker to consider all relevant aspects of the network. SIMULA' SIMUlation LAnguage' represents an effort to meet this need with regard to discrete-event networks ...*[16]

Die bahnbrechende Bedeutung von SIMULA liegt in der Einführung der als generisch objektorientiert angesehenen Konzepte 'Klasse/Objekt', Vererbung und dynamisches Binden.

Ziel des Abschnittes 3.2.1 ist zunächst die Bestimmung des Begriffs *Objektorientierung*. Aus der Vielzahl der unterschiedlichen Interpretationen und Definitionen dieses Begriffs wird hier exemplarisch ein pragmatischer, ingenienieurwissenschaftlicher Deutungsansatz, der vor allem in der Prozessleittechnik Verwendung findet, ausgewählt und erläutert. In den darauffolgenden Kapiteln 3.2.2 und 3.2.3 werden die UML als derzeit wichtigstes Beschreibungsmittel für die objektorientierte Modellierung und die mit der UML verwandten Spezifikationen eingehend beschrieben und verglichen.

3.2.1 Konzepte der Objektorientierung

Mit dem Begriff *Objektorientierung* wird zunächst eine Reihe spezieller programmiersprachlicher Konzepte bezeichnet. Sprachen, die ein Mindestmaß dieser Konzepte erfüllen, werden objektorientiert genannt. Jedoch herrschen in der Informatik unterschiedliche Auffassungen darüber, welche Konzepte die Objektorientierung auszeichnen. Die Diskussion, was die Objektorientierung auszeichnet, ist noch längst nicht beendet und wird immer noch kontrovers geführt. Drastisches Beispiel hierfür ist der im Informatik-Spektrum geführt Diskurs zwischen Broy [Broy 2002] und Jähnichen [Jähnichen 2002]. Weitere

[16] nach [Holmevik 1995]

wichtige Begriffsbestimmungen für die Objektorientierung finden sich z.B. in [Witt 1992]: *Objektorientierung = abstrakter Datentyp + Vererbung + Indentität.*

Neben der reinen Software-Entwicklung hat auch das Wissensmanagement bzw. die Wissensmodellierung von objektorientierten Konzepten profitiert. Hier sind vorallem die statischen Strukturierungskonzepte der Objektorientierung zu nennen:

- Klassen (Ausprägungsmuster für Objekte) mit Attributen und Methoden,
- Objektidentität
- Vererbungsmechanismen zwischen Klassen

All diese Konzepte, insbesondere die Vererbung, haben natürlich auch dynamische Aspekte (z.B. Polymorphie, dynamisches Binden), die hier aber außer Betracht gelassen werden können.

Eine wichtige ingenieurwissenschaftliche Anwendung für die statischen, objektorientierten Konzepte zur Informationsmodellierung findet sich in der Prozessleittechnik für verfahrenstechnische Anlagen.

In [Arnold 1998] wird beschrieben, welche objektorientierten Modellerierungskonzepte und in welcher Reihenfolge diese bei der Analyse leittechnischer Problemstellungen zur Anwendung kommen sollten: Zu Beginn erfolgt die Identifizierung der Objekte. Danach folgt die Attributierung und die Bestimmung der Relationen, die zwischen den Objekten herrschen. Letzteres kann auch als *Relationieren* bezeichnet werden, siehe [Polke 1993]. Neben den anwendungsspezifischen Assoziationen, die zwischen beliebigen Objekten herrschen können, sind insbesondere die *'besteht-aus'*-Beziehungen (Aggregation, Komposition) und die *'ist-ein'*-Beziehungen (Generalisierung, Spezialisierung) von besonderer Bedeutung. Nach [Ahrens 1994] manifestieren sich in diesen Beziehungen zwei grundlegende Architekturprinzipien der Informationsstrukturierung, nämlich die komplexbildende Abstraktion bei *'besteht-aus'*-Beziehungen und die klassenbildende Abstraktion bei *'ist-ein'*-Beziehungen. Ein weiteres wichtiges Architekturprinzip ist die Zerlegung. Das Zerlegungsprinzip findet sich in der Objektorientierung beim Identifizieren und Attributieren von Objekten.

Die folgende Tabelle 3.1 fasst diese Architekturprinzipien und die für die Informationsmodellierung notwendigen objektorientierten Konzepte zusammen. Weiterhin zeigt die Tabelle im Vorgriff auf den Abschnitt 3.2.2 die entsprechenden UML-Modellelemente und -Diagrammtypen und stellt dar, in welcher Reihenfolge die Strukturierungskonzepte der Objektorientierung beim *top-down*-Entwurf und beim *bottom-up*-Entwurf durchlaufen werden.

3.2.2 UML - *Unified Modeling Language*

Neben der reinen programmiersprachlichen Implementierung bietet die Objektorientierung auch die Möglichkeit der implementierungsunabhängigen Beschreibung und Spezifikation von Systemen (auch über den softwaretechnischen Bereich hinaus) - so, wie dies

Architekturprinzip	***Zerlegung***		***Abstraktion***	
			komplexbildend	klassenbildend
OO-Konzept	Identifizierung, Instanzierung	Attributierung, Relationieren (anwendungsspez. Relationen)	Relationieren (*besteht_aus*)	Relationieren (*ist_ein*), Klassifizieren, Vererbung
Vorgehen	down ◄ top			
	bottom ► up			
UML Modellierungselement	Objekt	Attribut, Assoziation	Aggregation, Komposition	Klasse, Generalisierung
UML Diagrammtyp	Objektdiagramm			
		Klassendiagramm		

Tabelle 3.1: Architekturprinzipien der Informationsstrukturierung und objektorientierte Konzepte

von den Erfindern von SIMULA geplant war (siehe oben). Seit der Mitte der 1990er Jahre hat sich hierbei die *UML* (Unified Modeling Language) als das führende Sortiment grafischer Beschreibungsmittel zur objektorientierten Spezifikation herauskristallisiert. Die Zielstellung dieser Sprache geht aus folgendem Zitat hervor:

> *The Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system.*[17]

Bevor wir näher auf den Aufbau von UML und die einzelnen in ihr enthaltenen Beschreibungsmittel eingehen, wird ein kurzer Abriss zur Geschichte objektorientierter Beschreibungsmittel und Methoden gegeben.

3.2.2.1 Rückblick

Ende der 80er und Anfang der 90er Jahre des 20. Jahrhunderts wurde eine Vielzahl grafischer Beschreibungsmittel zur objektorientierten Systemspezifikation vorgestellt, die im Gegensatz zu Programmiersprachen leicht zu erlernen und aufgrund der grafischen Repräsentation anschaulich und intuitiv zu verwenden sind, aber auch oftmals keine formale Semantik besitzen. Positiv zu verzeichnen ist, dass einige Autoren neben den reinen Beschreibungsmitteln auch methodische Vorgehensweisen vorschlugen, wie diese Beschreibungsmittel während der Systemanalyse einzusetzen sind. Zu deren wichtigsten Vertretern zählen die folgenden drei Methoden. Sie fanden am meisten Verbreitung und ihre Autoren sind die Protagonisten der UML-Entwicklung.

- OOD (*Object Oriented Design*), [Booch 1991] - OOD bietet unterschiedliche Diagramme, um damit auch unterschiedliche Sichten auf ein System zu ermöglichen. Zur strukturellen Modellierung bietet OOD Klassendiagramme, Objektdiagramme

[17] [OMG 2003a], S. XXV

und Moduldiagramme. Für die Verhaltensmodellierung stehen Zustands/Übergangs-Diagramme, Timing- und Prozessdiagramme zur Verfügung. Als Vorgehensmodell zum objektorientierten Design nennt Booch das *'Round-Trip Gestalt (sic!) Design'*.

- OMT (*Object Modeling Technique*), [Rumbaugh et al. 1993] - OMT unterscheidet Objektmodelle für statische, strukturelle Aspekte, dynamische Modelle für zeitliche Verhaltensaspekte und funktionale Modelle für Übergangs- und Funktionsaspekte. Die dazugehörigen Beschreibungsmittel sind Objektdiagramme (mit Klassen, Objekten und Relationen), strukturierte Zustandsdiagramme nach [Harel 1987] und Datenflußdiagramme. Methodisch unterscheidet Rumbaugh die Analysephase mit dem Analysemodell und die Entwurfsphase mit dem Entwurfsmodell, das im Gegensatz zum Analysemodell implementierungsabhängig ist. Sowohl die Analyse- als auch die Enwurfsphase sind in iterative Teilphasen untergliedert. Die Objektmodelle sollen vor den Verhaltens- und Funktionsmodellen erstellt werden.
- OOSE (*Object Oriented Software Engineering*), [Jacobson 1992]) - Jacobson unterscheidet beim Entwurf objektorientierter Software folgende Phasen: Anforderungserstellung, Analyse, Design, Implementierung und Test. Zur Systemspezifikation innerhalb der Phasen dienen Objektdiagramme, *use-case*-Diagramme, Analysediagramme, Interaktions- und Zustandsübergangsdiagramme.

Die genannten Methoden haben unterschiedliche Stärken und Schwächen. Während sich OOD durch gute Möglichkeiten zur Beschreibung von Echtzeitsystemen (durch Timing-Diagramme) und zur Zerlegung/Darstellung von Software in Komponenten und Module auszeichnet, eignet sich OMT mit seinen aussagekräftigen Klassendiagrammen für die Modellierung datenintensiver Informationssysteme. Das hervorragende Merkmal von OOSE ist die anwendungsfallorientierte Modellierung mit *use-cases*. Um diese unterschiedlichen Modellierungsmöglichkeiten miteinander zu vereinen, starteten zunächst Booch und Rumbaugh und wenig später noch Jacobson die Initiative zur *Unified Modeling Language* und stellten 1996 die Version 0.9 der UML vor. Ein Jahr später folgte unter der Mitarbeit führender IT-Unternehmen wie DEC, HP, IBM, TI usw. die Version 1.0 als offizieller OMG[18]-Standard. Seit dieser Zeit wird die UML unter Leitung der OMG als firmen- und herstellerunabhängiger Modellierungsstandard weiterentwickelt.

3.2.2.2 Grafische Notation

Wichtiger und wohl auch bekanntester Bestandteil der UML-Spezifikation sind die grafischen Beschreibungsmittel bzw. Notationen, mit denen die unterschiedlichen Systemaspekte modelliert werden können. Die UML vereint insgesamt neun solcher Beschreibungsmittel. Die mit den Beschreibungsmitteln repräsentierbaren Diagramme/Modelle lassen sich nach [Jacobson 1998] in Diagramme zur Strukturmodellierung und zur Verhaltensmodellierung einteilen. Im einzelnen sind dies die im folgenden genannten Diagrammtypen:

[18] *Object Management Group*

- Strukturmodellierung
  Klassendiagramm, Objektdiagramm, Komponentendiagramm, *Deployment*-Diagramm [19]

- Verhaltensmodellierung
  use-case-Diagramm [20], Sequenzdiagramm, Kollaborationsdiagramm, Zustandsdiagramm, Aktivitätsdiagramm

Jedes Diagramm wiederum besteht aus einer Reihe verschiedener Modellelemente, die im Allgemeinen nicht exklusiv in nur einem Diagrammtype verwendbar sind. Die Tabelle 3.2 auf Seite 70 zeigt die wichtigsten Modellelemente und die Diagramm-Typen, in denen sie vorzugsweise Verwendung finden. Diese Zusammenstellung erhebt keinen Anspruch auf Vollständigkeit und beruht im weitesten auf [Jacobson 1998] und [Oestereich 2001].

Auf die Gestalt/Form der einzelnen Modellelemente wird hier im einzelnen nicht näher eingegangen, da diese hinlänglich dokumentiert und bekannt sind.

3.2.2.3 Syntax und Semantik

Die Definiton einer Programmiersprache stützt sich auf die Bereitstellung der Komponenten Syntax und Semantik [Ghezzi 1989]. Die Syntax ist - vergleichbar mit der Grammatik einer natürlichen Sprache - für die Gültigkeit bzw. Zulässigkeit von Ausdrücken verantwortlich. Die Semantik legt die Bedeutung bzw. Wirkung gültiger Ausdrücke fest. Während sich für die Syntaxspezifikation formale Notationen auf der Basis kontextfreier Grammatiken wie z.B. die Backus-Naur-Form durchgesetzt und bewährt haben[21], gibt es noch keine allgemein akzeptierte Methode zur formalen Beschreibung der Semantik einer Sprache. Mögliche Ansätze sind operationale, axiomatische oder denotationale Semantik.

Die UML-Spezifikation bemüht sich ebenfalls um eine klare Syntax- und Semantikdefinition. Dazu wird nach dem Muster der ontologischen Wissensrepräsentation die UML selbst als Gegenstandsbereich angesehen und mit Hilfe einer UML-Ontologie durch Metaisierung konzeptualisiert. Diese UML-Ontologie wird deshalb auch als UML-Metamodel oder *Object Analysis & Design Facility Metamodel* (OA&DF Metamodel) bezeichnet. Die herausragende Besonderheit dieses Metamodels ist jedoch, dass es mit der UML selbst beschrieben wird - das Metamodel ist also selbstreferenzierend. Diese Selbstreferenzierung wird anhand eines kleinen Ausschnittes aus dem UML-Metamodell [OMG 2003a] in der Abbildung 3.8 auf Seite 71 deutlich.

Das gezeigte Partialmodell spezifiziert das Klassenkonzept (Metaklasse Class) und die Generalisierungs- und die Assoziationsbeziehung (Metaklassen Generalization, Association). Die zur Beschreibung dieses Partialmodells verwendeten Modellelement (Klassen,

[19] Verteilungs- oder Verwendungsdiagramm
[20] Anwendungsfall-Diagramm
[21] Die Syntax von Programmiersprachen hat durchaus kontextsensitive Anforderungen, die mit BNF nicht dargestellt werden können, siehe [Ghezzi 1989], S. 76

Modellelement	Diagrammtyp									
	statisch				dynamisch					
	Klassen~	Objekt~	Kompo-nenten~	Verteilungs~	use-case~	Sequenz~	Kollabo-rations~	Zustands~	Aktivitäts~
Klasse	x		[x]						
Interface	x		x						
Kollaboration					x				
use-case					x				
Komponente			x	[x]					
Knoten				x					
Objekt	[x]	x	[x]	[x]		x	x		x
Akteur					x	x			
Nachricht						x	x		
Lebenszyklus/ Zeitachse						x			
Paket	x		[x]	[x]	[x]				
Abhängigkeit	x		x	x	x		x		x
Assoziation	x		x	x	x				
Generalisierung	x		x		x				
Realisierung	x		x		x				
Link		x							
Zustand								x	x
Übergang								x	x
Ereignis								x	
Aktion								x	
Synchronisation									x
Verzweigung									x

Tabelle 3.2: Wichtige Modellelemente der UML und ihre Verwendung in den unterschiedlichen Diagrammtypen

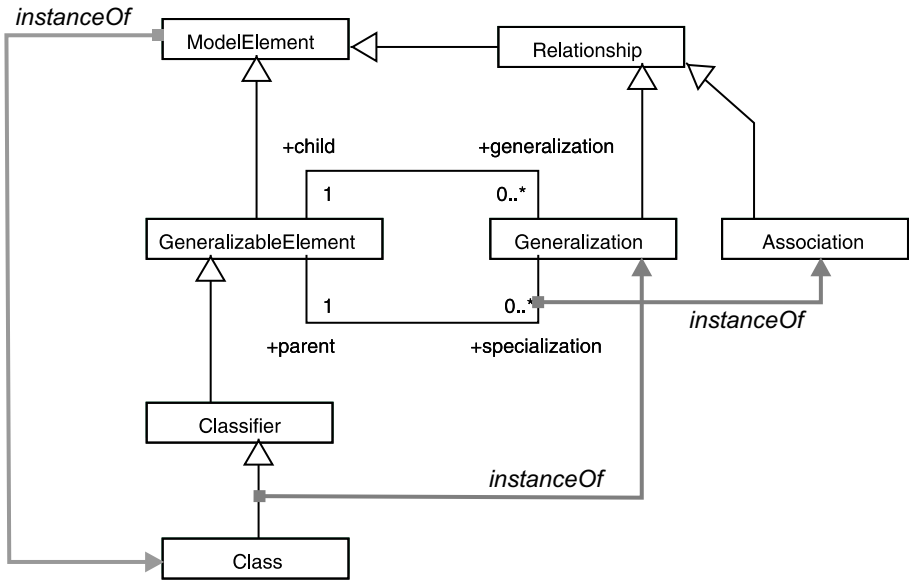


Abbildung 3.8: Selbstreferenzierendes Partialmodell aus dem UML Metamodell

Generalisierungen und Assoziationen) können allesamt als Instanzen dieser Metaklassen interpretiert werden, und dadurch ist dieses Partialmodell selbstbeschreibend.

Alle in der UML verwendeten Modellelemente und -konzepte werden im Metamodell als Metaklassen modelliert. Diese Metaklassen sind taxonomisch geordnet und durch syntaktische und semantische Relationen zueinander in Beziehung gesetzt. Eine eindeutige Trennung von Syntax und Semantik ist somit nicht mehr möglich. Ein konkretes UML-Diagramm besteht dann aus Instanzen dieser Metamodell-Klassen (strikte Metamodellierung) und ist ontologisch gesehen ein intendiertes Modell der UML-Ontologie.

Die folgende Abbildung 3.9 zeigt den Aufbau des UML-Metamodells, das in mehrere Pakete hierarchisch untergliedert ist, zwischen denen Abhängigkeiten bestehen.

Ein Teil des Metamodells ist im Foundation-Paket zusammengefasst. Dieses wiederum besteht im Wesentlichen aus dem Core-Paket mit den Metaklassen für die wichtigsten statischen Modellelemente (u. a. Klassen, Relationen, strukturelle und verhaltensbestimmende Eigenschaften wie Attribute und Methoden) und den Metaklassen für das Namensraum-Konzept und für Abhängigkeiten zwischen Modellelementen. Das DataType-Paket stellt die UML-Basistypen (*String*, *Integer*, ...) und Aufzählungstypen, die für die UML-Definition notwendig sind (*AggregationKind*, *VisibilityKind*, ...), zur Verfügung.

Der andere Teil des Metamodells liegt im Behavioral Elements-Paket und dient der Spezifikation der verhaltensbestimmenden Beschreibungsmittel. Kollaborations- und Sequenzdiagramme werden gemeinsam durch Metaklassen aus dem Collaborations-Paket konzeptualisiert. Die Spezifikationen der restlichen verhaltensbestimmenden Diagrammtypen - *use-cases*, Zustands- und Aktivitätsdiagramme - finden sich in den entsprechend benannten Pakten.

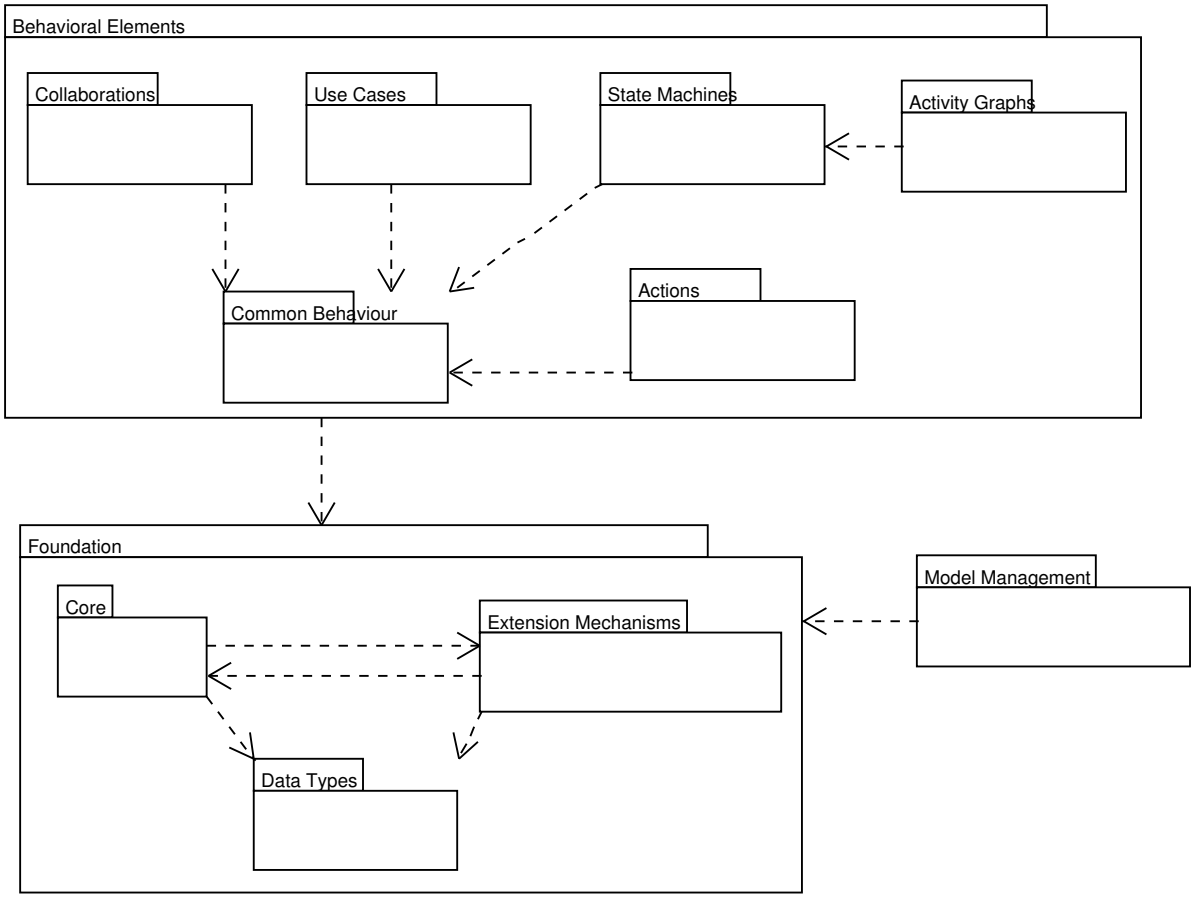


Abbildung 3.9: Aufbau des UML Metamodels

3.2.2.4 OCL - *Object Constraint Language*

Die Praxis zeigt, dass anwendungsspezifisches Wissen mit der UML allein nicht genau genug spezifiziert werden kann. Häufig tritt der Fall auf, dass die durch ein Klassendiagramm approximierten intendierten Modelle weiter eingeschränkt werden müssen. Hierfür wird in [OMG 2003a] die *Object Constraint Language* (OCL) spezifiziert. OCL ist eine formale Sprache, mit der *constraints* zur Einschränkung von UML-Modellen formuliert werden können und die vollständig in das UML-Metamodell integriert ist (siehe Abbildung 3.10).

Ein OCL-*contraint* ist ein Objekt der Klasse Constraint und erbt von der Klasse ModelElement sein Namensattribut. Der eigentliche OCL-Ausdruck eines *constraint* ist ein Objekt der Klasse Expression und trägt das language-Attribut 'OCL'. Ein *constraint* kann vom Typ *Invariante*, *Vorbedingung* und *Nachbedingung* sein. Dieser Typus wird durch die Schlüsselwörter *inv*, *pre* oder *post*, die Objekte vom Typ Stereotype sind, festgelegt. Die Syntax von OCL ist ebenfalls Teil der UML Spezifikation und ist dort als kontextfreie Grammatik in EBNF definiert. Danach ist ein OCL-*constraint* folgendermaßen aufgebaut. Anfangs steht eine obligatorische Kontextdeklaration, in der bestimmt

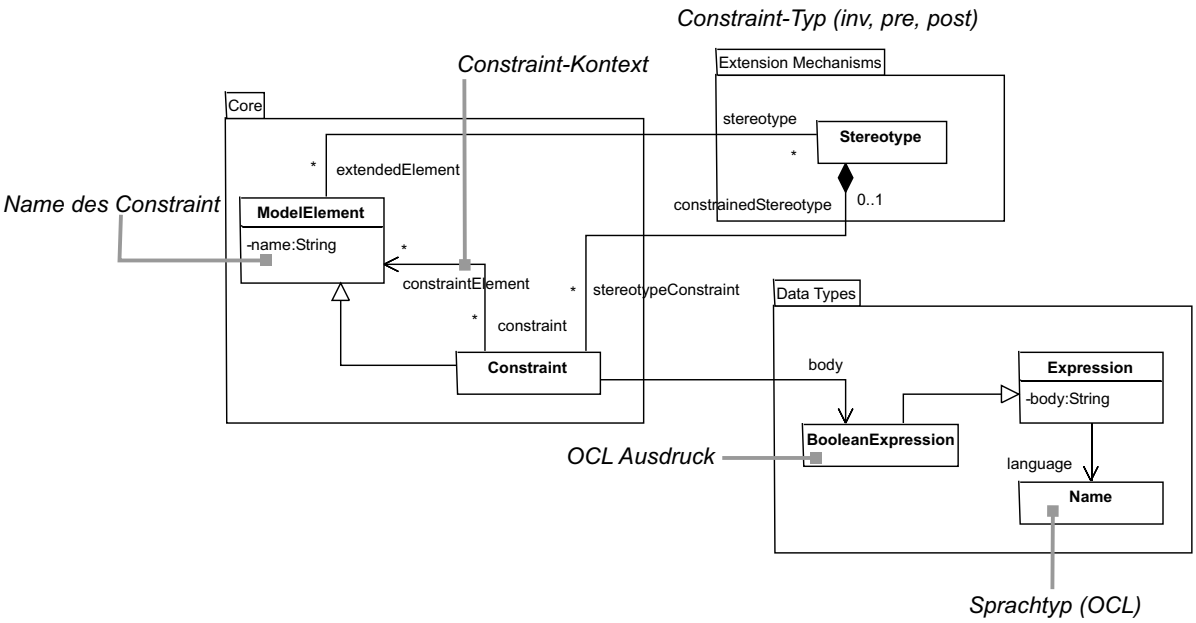


Abbildung 3.10: *Constraint*-Spezifikation im UML Metamodell nach [OMG 2003a]

wird, für welche Klasse eines gegebenen UML-Modells der *constraint* gilt. In den darauf folgenden Schlüsselwörtern wird spezifiziert, ob es sich um eine Invariante oder eine Vor-/Nachbedingung für eine Methode der im Kontext spezifizierten Klasse handelt. Danach erst folgen die eigentlichen logischen Ausdrücke, die im weitesten mit prädikatenlogischen Ausdrücken erster Stufe vergleichbar sind und hier nicht weiter erläutert werden sollen. Über die Entscheidbarkeit und Berechenbarkeit von OCL-Ausdrücken werden in [OMG 2003a] keine weiteren Aussagen getroffen.

OCL dient wie ja bereits oben erwähnt der Einschränkung der intendierter Modelle einer mit UML formulierten Ontologie. Dies soll am bereits erwähnten Blockmodell auf Seite 44 erläutert werden. Hier werden Blöcke als Instanzen einer Klasse Block modelliert. Die Relationen zwischen den Blöcken entsprechen den Assoziationen im Klassendiagramm. Ein mögliches Klassendiagramm mit den Assoziationen steht_auf und steht_unter zeigt Abbildung 3.11.

Durch die Kardinalitäten wird klar, dass ein Block genau auf einem anderen oder auf keinem stehen kann, während ein Block unter keinem oder beliebig vielen stehen kann. Die genaue Semantik dieser Relationen geht aber aus diesem Klassendiagramm nicht hervor, da es sich bei diesen Relationen um konzeptuale Relationen handelt, die je nach Objektkonstellation andere Objekte miteinander verknüpfen. Weiterhin läßt das Klassendiagramm Objektkonstellationen zu, die nicht sinnvoll sind (ein Block kann auf sich selbst stehen). Um die mit dem Klassendiagramm intendierten Objektmodelle dahingehend einzuschränken, dienen die folgenden *constraints*:

```
context Block
inv first:
```

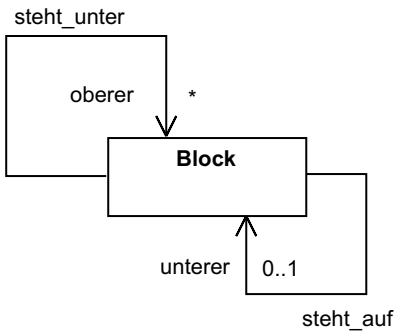


Abbildung 3.11: Klassendiagramm für Blöcke

```
   Block.allInstances->forAll(a, b, c |
      b = a.unterer and c = b.unterer
         implies
      c.oberer->contains(a)
inv second:
   (self.unterer <> self)
      and
   (self.oberer->excludes(self))
```

Während die erste Invariante durch Implikationen die Bedeutung der unterer- und oberer-Relationen weiter klärt, schließt die zweite Invariante die absurden Fälle aus, dass ein Block über oder unter sich selbst stehen kann und schränkt damit die möglichen Objektmodelle des Klassendiagramms weiter ein.

Wie bereits erwähnt dient OCL der Einschränkung der intendierten Modelle einer mit UML formulierten Ontologie und ermöglicht so, eine Konzeptualisierung genauer zu approximieren, als dies ohne OCL möglich wäre. In der vorliegenden Arbeit wird OCL hauptsächlich dazu genutzt, die in Kapitel 5 präsentierte Strukturverträglichkeit automatisierungstechnischer Ontologien zu definieren.

3.2.2.5 Entwurfsmethodik

Während die oben erwähnten Vorläufer der UML neben den reinen Beschreibungsmitteln auch die passenden objektorientierten Entwurfsmethoden liefern, ist die Entwurfsmethodik ausdrücklich kein Bestandteil der UML-Spezifikation [OMG 2003a], da die UML prozess- bzw. methodenunabhängig sein soll und ihre Beschreibungsmittel verschiedene Entwufsprozesse unterstützt. Darüber hinaus sind Entwurfsprozesse stark unternehmens- und projektabhängig, so dass eigentlich kein standardisierter Prozess spezifiziert werden kann. Trotz allem wird in [OMG 2003a] folgende Empfehlung ausgesprochen:

Although the UML does not mandate a process, its developers have recognized

> *the value of a use-case driven, architecture-centric, iterative, and incremental process, so were careful to enable (but not require) this with the UML.*[22]

Die hier entstehende Standardisierungslücke wird von Softwarefirmen genutzt, die die unterschiedlichsten UML-Vorgehensmodellen und Entwurfsmethoden anbieten. Als prominentes Beispiel sei hier an dieser Stelle der *Rational Unified Process* von der Firma *Rational* genannt. Dieser Prozess geht auf Booch's Entwurfsmethodik zurück und wurde bei der Firma Rational von Booch, Rumbaugh und Jacobson weiterentwickelt [Kruchten 1999].

3.2.3 UML und verwandte Spezifikationen - OMA (*Object Management Architecture*)

Die *Object Management Group* hat eine Reihe von Standards für die Modellierung objektorientierter Softwaresysteme veröfffentlich, die als *Object Management Architecture* (OMA) bezeichent werden. Neben der UML enthält die OMA die Spezifikationen von MOF (*Meta Object Facility*, [OMG 2002]) und XMI (*XML Metadata Interchange*, [OMG 2003b]). Beide Spezifikationen hängen eng mit der UML zusammen und werden im folgenden kurz erläutert.

MOF - *Meta Object Facility*

Wie im Abschnitt 3.2.2.3 bereits erwähnt, ist die Syntax und Semantik der UML in einem selbstreferenzierenden Metamodell festgehalten, dem OA&DF Metamodell . Bei der OMG wurde bald die Notwendigkeit einer allgemeingültigen Methode zur Modellierung solcher Metamodelle erkannt und deshalb MOF entwickelt. MOF ist eine objektorientierte Sprache zur Modellierung von Metamodellen, wie sie z.B. bei der UML-Spezifikation benötigt werden, und sehr ähnlich zu UML. MOF selbst wird durch ein Meta-Metamodell beschrieben, das dem UML-Metamodell sehr ähnelt und auch selbstreferenzierend und damit selbstbeschreibend ist. Da die Metamodellarchitekturen von MOF und UML sehr ähnlich sind, kann man MOF-Modelle mit der UML-Notation repräsentieren. Aufgrund der großen Ähnlichkeiten ist die OMG bemüht, beide Spezifikationen einander anzugleichen.

Der auffälligste Unterschied zwischen MOF und UML ist der, dass sie auf unterschiedlichen Metaebenen angesiedelt sind. MOF ist eine *top-level*-Ontologie und befindet sich eine Ebene höher als das UML-Metamodell. Weiterhin gibt es wichtige Modellierungsunterschiede: MOF kennt im Gegensatz zu UML z.B. keine Assoziationsklassen, keine Interfaces, keine mehrwertigen (*n-ary*) Attribute und keine Attributqualifizierung[23]. Bei MOF hingegen ist die Modellierung von Objektreferenzen möglich. UML kennt nur Assoziationen oder Attribute. Unterschiede bestehen auch in der Aggregationssemantik und bei den Datentypen.

[22] [OMG 2003a], Kap. 1.5.13
[23] [Jacobson 1998], S. 147

XMI - *XML Metadata Interchange*

Während MOF zur Spezifikation von Metamodellen entwickelt wurde, dient XMI der Repräsentation von MOF-Modellen in Form von XML-Dateien, wobei der Abstraktionsgrad dieser MOF-Modelle nicht relevant ist. Das heißt, dass mit MOF neben Metamodellen auch ganz konkrete Datenmodelle modelliert und die entsprechenden Instanzen anschließend per XMI als XML-Dateien gespeichert werden können, [Grose 2002]. XMI unterstützt in der aktuellen Version 2.0 nur noch den XML-Schema-Standard [W3C 2000] als Spezifikation des XML-Dateiformates. DTD wird nicht mehr unterstützt. Die XMI-Spezifikation gliedert sich demnach auch in zwei Teile, die *XML Schema Production* und die *XML Document Production*. Während im erstgenannten Teil die Erstellung von XML-Schema-Dateien aus einem MOF-Datenmodell spezifiziert wird, beschreibt der zweite Teil, wie Instanzen des MOF-Datenmodells als Anwenderdaten in Form von XML-Dateien, die mit dem vorher erzeugten XML-Schema konform sind, abgelegt werden können.

Gerade bei der Anwendung von XMI ist es wichtig, die Eigenheiten von MOF und dessen Unterschiede zur UML zu beachten, da die oben erwähnten Aggregations- und Referenzsemantiken von MOF auf spezielle XML-Konstrukte abgebildet werden. XMI und MOF werden in dieser Arbeit dazu genutzt, automatisierungstechnische Ontologien zu definieren und Verhaltens- und Strukturmodelle auszutauschen.

3.3 Zusammenfassung

Ziel des Kapitels war die Darstellung ausgewählter Methoden und Verfahren zur Gewinnung und Formalisierung von deklarativem Anwendungswissen. Die mit diesen Methoden generierten Modelle werden Informationsmodelle genannt, der Prozess Informationsmodellierung. Neben den Ontologien wurden hier objektorientierte Methoden näher vorgestellt.

Der Begriff Ontologie stammt ursprünglich aus der Philosophie und hat in die Wissensverarbeitung Einzug gehalten, wobei hier unterschiedliche, mehr oder weniger formale Definitionen zu finden sind. Neben der Definition von Gruber wurde die formale Ontologiedefinition nach Guarino detailliert dargestellt. Danach ist eine Ontologie ein System von logischen Aussagen, um die intendierten Modelle einer Konzeptualisierung approximieren zu können.

Desweiteren wurden grundlegende ontologische Paradigmen identifiziert (Konzpetualisierung, Taxonomisierung, Vererbung, Instanzierung) und zwei Möglichkeiten zur Ontologieintegration (Metaisierung, Strukturverträglichkeit) vorgestellt. Die Strukturverträglichkeit ist in dieser Arbeit das grundlegende Prinzip für die Integration der später vorgestellten automatisierungstechnischen Beschreibungsmittel und Referenzmodelle.

Wie in Abschnitt 3.2.1 gezeigt hängen objektorientierte Methoden eng mit ontologischen Methoden zusammen, da beide teilweise auf ähnlichen Paradigmen beruhen. Als

prominentestes objektorientiertes Beschreibungsmittel wurde dann die UML aus historischer Sicht und deren Syntax und Semantik anhand des UML-Metamodells (OA&DF) erläutert.

Ein wichtiger Bestandteil der UML ist aus ontologischer Sicht die OCL, da diese die effektive Approximation bzw. Einschränkung der intendierten Modelle einer gegebenen Ontologie ermöglicht.

Den Abschluss diese Kapitels bildete dann die Darstellung ausgewählter, UML-verwandter OMA-Spezifikationen, da diese bei der Implementierung der am Ende dieser Arbeit vorgestellten XML-basierten Werkzeugumgebung Verwendung finden.

4 Ausgewählte Beschreibungsmittel und Referenzmodelle der Automatisierungstechnik

Da die Automatisierungstechnik als ausgesprochen interdisziplinäre Wissenschaft auf die Beschreibung von sowohl energetischen als auch informatischen Prozessen angewiesen ist, kommt eine Vielzahl unterschiedlicher Beschreibungsmittel wie z.B. Blockschaltbilder, Zustandsautomaten, verschiedene Petrinetztypen, Datenflussdiagramme, Funktionsbausteine, Bondgraphen usw. zum Einsatz. Um diese Beschreibungsmittel der weiteren informationstechnischen Verarbeitung (Speicherung, Analyse und Synthese von Modellen) zugängliche zu machen, können diese mit den im vorangegangen Kapitel 3 vorgestellten Konzepten zur Informationsmodellierung in Form von Ontologien konzeptualisiert und damit formalisiert werden. Solche Ontologien deuten auf das dem Beschreibungsmittel zugrundeliegende Modellkonzept hin und bergen wie in [Schnieder 2001] gezeigt enormes Nutzungspotenzial.

Zu den für die Automatisierungstechnik relevanten Ontologien zählen aber auch spezielle Informationsmodelle mit normativem Charakter (sogenannte Referenzmodelle), die bereits in konzeptualisierter Form vorliegen. Es handelt sich hier um kontextabhängige und mehr oder weniger abstrakte/konkrete Ontologien, die meist zur Modellierung struktureller, statischer Aspekte in Automatisierungssystemen dienen. Prominentes Beispiel ist das integrierte Produktmodell von ISO1303, das in Abschnitt 4.3 näher erläutert wird. Weitere produktdefinierende Ontologien sind z.B. *e@Class* ([Eibl et al. 2000]) und *CAEX* ([Drath 2004]).

Ausgehend von der Problemstellung dieser Arbeit, strukturelle Ähnlichkeiten zwischen verhaltensbeschreibenden und strukturbeschreibenden Ontologien zu finden und diese im Modellierungsprozess aufrechtzuerhalten (strukturtreue Modellbildung), werden zunächst einmal die Beschreibungsmittel Petrinetze und Bondgraphen vorgestellt. Mit ihnen kann man das Verhalten energetischer/kontinuierlicher (Bondgraphen) bzw. informatischer/diskreter Prozesse (Petrinetze) modellieren. Beide Beschreibungsmittel beinhalten aber auch eine konzeptbedingte Struktursicht, die mit dem struktur- bzw. gerätebeschreibenden Produktmodell nach ISO 10303 abgeglichen werden kann. Somit können aus diesen Beschreibungsmitteln strukturverträgliche Ontologien konstruiert werden, siehe Kapitel 5.

Den Abschluss des Kapitels bildet dann die formale Prozessbeschreibung nach VDI/VDE 3682. Hier handelt es sich um einen Beschreibungsmittelansatz zur Integration prozess-

und informatikorientierter Sichten, der sich ebenfalls zum Aufsetzen eines übergeordneten strukturtreuen Modellbildungsprozesses eignet. Dies würde jedoch den Rahmen der vorliegenden Dissertation sprengen und wird hier nicht weiter verfolgt.

4.1 Petrinetze

Wichtige Impulse erfuhr die Netztheorie seit Anfang der 60er Jahre des letzten Jahrhunderts durch Carl Adam Petri, der mit seiner 1962 vorgelegten Dissertation mit dem Titel *'Kommunikation mit Automaten'* ([Petri 1962]) das Beschreibungsmittel der Petrinetze (Aktionsnetze) begründet hat. Dieses Beschreibungsmittel wird im folgenden genauer vorgestellt, wobei hier die Argumentationskette von Petri, mit der er die Notwendigkeit neuer Beschreibungsmittel zur Darstellung von Kommunikationsprozessen erklärt, grob nachvollzogen wird. Dadurch wird Sinn und Zweck von Petrinetzen einsichtig.

4.1.1 Grundlagen

Petri bezeichnet die von ihm entwickelten Netze als *' ... ein Darstellungsmittel für komplizierte organisatorische Vorgänge beliebiger Art.'*[1] Eine grundlegende Problemstellung seiner Arbeit [Petri 1962] ist u.a. die Modellierung von informationsverarbeitenden Maschinen zur Lösung rekursiver Aufgaben. Rekursiv formulierte Aufgaben sind durch das Problem gekennzeichnet, dass der Speicherbedarf, der zur deren Lösung benötigt wird, erst zur Laufzeit bekannt ist. Petri argumentiert, dass es deshalb notwendig ist, den Speicher einer Rechenmaschine bei Bedarf unbeschränkt erweitern zu können. Rechenmaschinen, die als endlicher, deterministischer und synchroner Automat konzipiert sind, lassen sich aber nicht unbeschränkt und modular erweitern, ohne an ihr konstruktive Änderungen vornehmen zu müssen. Denn pro Rechentakt muss die Maschine physikalisch unterscheidbare Zustände annehmen. Dies erfordet bei gegebenen Signal- und Bearbeitungslaufzeiten der verwendeten Bauelemente eine Anpassung des Rechentaktes bzw. eine Neuauslegung wegen Änderung von Leitungslängen. Dies bedeutet zusammengefasst: *'Die Automatentheorie kann den physikalisch-reellen Informationsfluss bei der Lösung einer rekursiven Aufgabe nicht darstellen.'*[2] Gefordert ist also ein Beschreibungsmittel, mit dem es möglich ist, *'Informationsmaschinen strukturell so auszulegen, daß sie asynchron, in allen Teilen parallel arbeitend, beliebig erweiterungsfähig sind.'*[3]

Darüber hinaus lässt sich gegen die synchronen Automaten anführen, dass nach der Relativitätstheorie der Gleichlauf von Zeit in einem System prinzipiell nicht gegeben sein muss. Die der Automatentheorie zugrundeliegende Annahme, dass ein System sich zu einem bestimmten Zeitpunkt in einem gemeinsamen Zustand befindet, ist fiktiv. Somit

[1] [Petri 1962], S. 3
[2] [Petri 1962], S. 6
[3] [Petri 1962], S. 3

lassen sich also der Gleichlauf von Uhren und die Konstruktion von Informationsmaschinen zur Lösung rekursiver Aufgaben in Analogie zueinander als Kommunikationsproblem auffassen. Petri konzipierte darauf aufbauend seine Netztheorie, um Kommunikationsphänomene modellieren und analysieren zu können. Neben der Kommunikation zwischen Automaten schließt dies auch die Kommunikation zwischen Mensch und Automaten ein.[4] Im folgenden wird kurz diese Netztheorie erläutert.

Zur Darstellung von Kommunikationsvorgängen dienen idealisierte, logische Schaltelemente, von denen aber die Einhaltung folgender Prinzipien gefordert wird:

1. Quantisierungsprinzip (Existenz einer quantisierten Größe)
2. Erhaltungsprinzip (Energieerhaltung, Massenerhaltung)
3. Reaktionsprinzip (jeder Messvorgang ändert die Eigenschaften des beobachteten Objektes, zerstörendes Lesen)

Das erste Postulat wird dadurch erfüllt, dass sich in dem *'Wirkungsbereich'*[5] eines Schaltelementes nur diskrete Objekte (bits) aufhalten können und von dort aus vom Schaltelement konsumiert bzw. erzeugt werden können. Petri verallgemeinert die Schaltelemente zu *Knoten* bzw. zu *Aktionen* (Aktionen sind *'Ein-Aktions-Knoten'*[6]) und die Wirkungsbereiche zu *Stellen*. Aktionen und Stellen werden durch Kanten miteinander verbunden. Durch die Kanten wird auch der Richtungssinn der Aktionen festgelegt, d.h. welche Stellen zum Vorbereich und welche zum Nachbereich der Aktion gehören. Durch Eintreten einer Aktion werden alle Objekte im Vorbereich konsumiert und im Nachbereich erzeugt.

Zur Erfüllung des Postulates 2) müssen die Aktionen *konservativ* sein, d.h. die Anzahl der Objekte im Wirkungsbereich einer Aktion bleibt beim Schalten der Aktion konstant.

Postulat 3) bedeutet, dass die Aktionen *reaktiv* sein müssen, d.h. dass sich beim Eintreten der Aktion auf jedem einzelnen Wirkungsbereich die Objektanzahl verändert. Kein Wirkungsbereich bleibt beim Schalten unberührt.

Konservative, reaktive Aktionen, die darüber hinaus noch singulär sind (keine gemeinsamen Stellen im Vor- und Nachbereich) heissen dann *Transitionen*.

Petri beschränkt die sogenannten Aktionsnetze strukturell zunächst auf Stellen mit genau zwei Transitionen im Vor- und Nachbereich und auf Transitionen mit genau zwei Stellen im Vor- und Nachbereich. Mit dieser Struktur sind die Postulate 1), 2) und 3) erfüllt. Solche Netze heissen vollständige Aktionsnetze, siehe Abbildung 4.1.

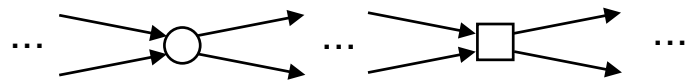


Abbildung 4.1: Struktur vollständiger Aktionsnetze

[4] vergleiche auch den Titel von [Petri 1962]
[5] [Petri 1962], S. 53
[6] [Petri 1962], S. 106

Netze, die diese Eigenschaften nicht besitzen, sind unvollständig und besitzen einen Rand. *'Alle in diesem Sinne unvollständig verknüpften Stellen und Aktionen bilden den Rand des Aktionsnetztes.'*[7] Nur unvollständige Aktionsnetze sind kommunikationsfähig: *'Da ein kommunikationsfähiges Netz seiner Natur nach unvollständig ist, sind wir ohnehin vorwiegend an unvollständigen Netzen interessiert.'*[8] Die Verknüpfung von unvollständigen Netzen zum Zweck der Kommunikation erfolgt über die Randstellen . Durch die Unvollständigkeit wird das Postulat der Erhaltung verletzt. Mircescu versucht in [Mircecsu 1997], das Erhaltungsproblem dadurch zu lösen, dass er eine fünfdimensionale Raumzeit mit drei Raumdimensionen, einer Zeitdimension und einer kausalen Dimension, deren Metrik durch den kausalen Abstand in *Petri* gemessen werden kann, einführt. Darauf aufbauend definiert er ganz axiomatisch einen erweiterten Erhaltungssatz. Trotz dieses Erhaltungssatzes bleibt jedoch die Notwendigkeit der Unvollständigkeit und damit Nichterfüllung der Erhaltung in kommunikativen Netze bestehen. Dieser Notwendigkeit wird aber bei [Mircecsu 1997] nicht Rechnung getragen.

Die eben beschriebene Art von Aktionsnetzen werden unabhängig von ihrer Vollständigkeit als *Stellen/Transitionsnetze* bezeichnet und ermöglichen die generische, anwendungsunabhängige Darstellung von Kommunikations- und Kooperationsprozessen. Diese Modellierungsmöglichkeiten bringen auch entscheidende Vorteile für den Entwurf von Automatisierungssystemen. Bereits modellierte Teilsysteme können dadurch kombiniert werden, indem die zwischen ihnen stattfindenden Kommunikationprozesse modelliert werden, ohne dass die Teilsysteme geändert oder ein ganz neues Gesamtmodell erstellt werden muss.

4.1.2 Automatisierungstechnische Interpretation von Petrinetzen

In [Petri 1962] wird gezeigt, wie informationsverarbeitende Systeme mit Netzen modelliert werden können. Bezüglich der (technischen) Realisierung von Netzmodellen konstatiert Petri selbst, *'daß die Wahl des Schaltelementes in §5 (Anmerkung: Transition) aufgrund physikalischer Überlegungen die Realisierung nicht erschwert, sondern ihr bei geeigneter Interpretation sogar entgegenkommt. Die weitere Erörterung dieser Fragen muß den Fachleuten überlassen bleiben'*. Tatsächlich existiert bereits eine Vielzahl von Netzrealisierungen bzw. -interpretationen[9], die den Transitionen und Stellen der Petrinetze bestimmte Elemente aus unterschiedlichen Gegenstandsbereichen wie z.B. Informatik, Verfahrenstechnik, Biologie usw. zuordnen und so die Petrinetze für die Modellierung in diesen Gegenstandsbereichen erst nutzbar machen. Wir betrachten nun einige der für die Automatisierungstechnik relevanten Netzinterpretationen.

[7] [Petri 1962], S. 114
[8] [Petri 1962], S. 111
[9] Realisierung bei der Systemsynthese, Interpretation bei der Systemanalyse

4.1.2.1 Steuerungsstechnik

Aus den in Abschnitt 4.1.1 erwähnten Erläuterungen ergibt sich zunächst einmal eine ganz naheliegende Netzinterpretation: Knoten sind elektronische/elektrische Bauelemente, ihre Anschlussstellen entsprechen den Wirkungsbereichen und werden durch Stellen repräsentiert. Die Verknüpfung der Knoten über gemeinsame Wirkungsbereiche entspricht der Verbindung von Schaltelementen durch Leitungen. Jedoch erschwert das von Transitionen eingehaltene Reaktionsprinzip die Modellierung von Datenverarbeitungsprozessen, da viele Bauelemente eben nicht reaktiv sind (d.h. die Objekte im Vorbereich werden nicht beeinflusst) bzw. die reaktiven Effekte vernachlässigt werden können wie z.B. bei einem lesenden Speicherzugriff. Ein weiteres Problem besteht in Konflikten, mit denen in Petrinetzen nicht-determinierte Prozesse (Abschnitt 2.3.3) modelliert werden können. Da die für die Steuerungstechnik typischen Schaltnetze im Gegensatz zu Schaltwerken jedoch immer determiniert arbeiten ([Fasol 1988]), sind die Konflikte in steuerungstechnischen Netzen auszuschließen. Die Abbildung 4.2 verdeutlicht die Problematik. Die Abbildung zeigt ein einfaches Schaltnetz und das dazugehörige in-

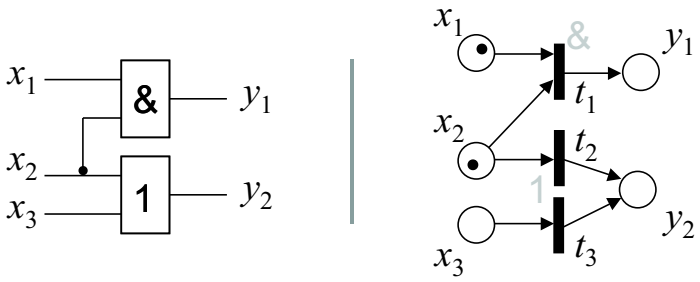


Abbildung 4.2: Konflikte in steuerungstechnischen Petrinetzen

terpretierte Netz. Diese einfache Netzinterpretation ist nur eingeschränkt zulässig, da die Transitionen beim Schalten die Marken der Eingangsplätze löschen und das Netz nicht-determiniert ist (Konflikt zwischen Transition t_1 und t_2). Jedoch wurden Ansätze entwickelt, diese Nachteile bei der Modellierung steuerungstechnischer Problemstellungen mit Petrinetzen zu überwinden:

- Analyse digitaler Schaltungen mit elementaren Petrinetzen ([König 1988])
  Bei diesem Modellierungsansatz werden Schaltelemente und signalführende Leitungen als B/E-Netze interpretiert, die mit Testkanten verknüpft sind und somit rückwirkungsfrei kommunizieren. Dieser Ansatz verursacht einen hohen Modellierungs- und Verknüpfungsaufwand, kommt aber Petris Grundidee der kommunizierenden Automaten sehr nahe.

- LS-Netze ([König 1988]):
  LS-Netze wurden zur Logiksimulation digitaler Schaltungen entwickelt. Hier werden wie zu Anfang von Kapitel 4.1.2.1 erwähnt die digitalen Bausteine als Transitionen und die Signalleitungen als Plätze interpretiert. Jeder Platz enthält soviele

Marken wie Transitionen in seinem Vorbereich. Des weiteren schalten bei diesen Netzen die Transitionen, sobald mindestens ein Platz im Vorbereich markiert ist. Diese Netze sind somit konfliktfrei. Der Nachbereich wird nur markiert, wenn der für eine Transition geltende logische Ausdruck je nach Markierung im Vorbereich wahr wird. Diese Netze ermöglichen zwar die völlig strukturtreue Umsetzung digitaler Schaltungen, besitzen aber eigene Schaltregeln.

- Speicherprogrammierbare Steuerungen

Neben den verbindungsorientierten Steuerungen können mit Petrinetzen auch speicherprogrammierbare Steuerungen (SPSen) modelliert werden ([v. Aspern 1994b], [v. Aspern 1994a], [Jörns 1995], [Chouikha 1998], [Ober 1999]). Die logischen Verknüpfungen in SPS-Programmen werden als Transitionen interpretiert werden; Eingänge, Ausgänge und Merker werden als Plätze interpretiert. Anstatt den lesenden Speicherzugriff auf Netzebene z.B. mit Testkanten zu modellieren, kann auf umgekehrte Weise der für Netze typische löschende Speicherzugriff im SPS-Programm implementiert werden. Die folgende Abbildung 4.3 zeigt ein einfaches Petrinetz und die dazu äquivalente Realisierung als SPS-Programm.

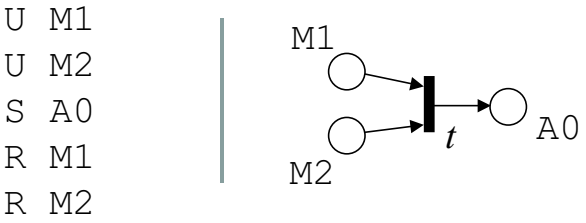


Abbildung 4.3: Petrinetze zur SPS-Programmierung

Hier wird das für Netze typische Abziehen der Marken im Vorbereich und das Belegen des Nachbereichs durch Setzen (`S`) bzw. Rücksetzen (`R`) der Merker- und Ausgangsspeicherstellen implementiert.

4.1.2.2 Verfahrenstechnik

Gegenstand der Verfahrenstechnik ist im weitesten Sinne die Herstellung chemischer Erzeugnisse. Dies geschieht meist in verfahrenstechnischen Fließprozessen, die sich von den fertigungstechnischen Stückprozessen[10] durch kontinuierliche Massenströme unterscheiden und mit prozessleittechnischen Einrichtungen gesteuert werden.

Von den unterschiedlichen Ansätzen, Petrinetze bei der Beschreibung verfahrenstechnischer Prozesse einzusetzen, wird hier beispielhaft die in [Hanisch 1992] beschriebene

[10] *batch*-Prozesse

Netzinterpretation vorgestellt. Hierbei handelt es sich um die verfahrenstechnische Interpretation für Kanal/Instanzen-Netze, B/E-Netze, S/T-Netze und P/T-Netze, mit der sowohl die verfahrenstechnischen Strecken (Systemanalyse) als auch die Steuerungsstrategien zur Lösung prozessleittechnischer Aufgabenstellungen (Steuerungsentwurf) beschrieben werden kann. Bei der Systemanalyse wird von der Annahme ausgegangen, dass verfahrenstechnische Produktionsprozesse aus Teilprozessen bestehen, die bei ihrem Ablauf Ressourcen benötigen. Ressourcen sind dabei Objekte wie Edukte, Produkte, Hilfsenergien, Katalysatoren, Apparate, Bedienpersonal usw., die vom Prozess verbraucht, erzeugt oder genutzt werden. Während sich die Prozesse durch Transitionen darstelllen lassen, entsprechen die Ressourcen bzw. deren diskrete Eigenschaften (binäre Eigenschaften bei B/E-Netzen, abzählbare/endliche Eigenschaften bei S/T-Netzen) den Stellen.

4.1.2.3 Agentensysteme

Eine weitere interessante Einsatzmöglichkeit von Petrinetzen ist die Modellierung von Agentensystemen. Hierbei handelt es sich um ein Forschungsfeld der künstlichen Intelligenz, dem in jüngster Zeit von Seiten der Informatik viel Interesse entgegengebracht wird. Zentraler Begriff ist der Agent als kooperatives System, das mit der Umwelt bzw. anderen Agenten kooperiert und kommuniziert. Wie in [Burkhard 2003] erläutert wird, kommt damit bei der Modellierung von Agentensystemen der Beschreibung verteilter, nebenläufiger Kooperations-/Kommunikationprozesse eine ganz besondere Bedeutung zu. Weiterhin ist die Beschreibung nicht-determinierter Prozesse notwendig, um z.B. Unsicherheiten eines Agenten über den aktuellen Systemzustand, die Umwelt und die zukünftige Entwicklung oder um Verhaltensvarianten zu modellieren. Aus den oben ge-

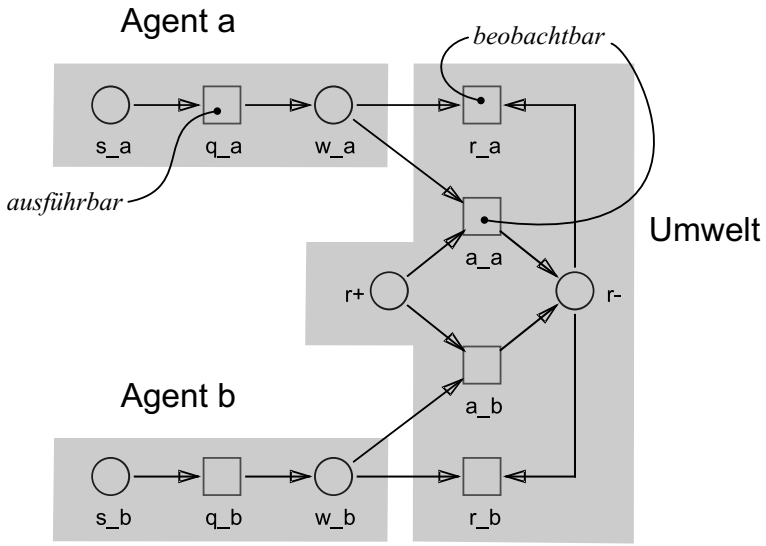


Abbildung 4.4: 2-Agenten-System nach [Burkhard 2003]

nannten Gründen bieten sich deshalb Petrinetze bei der Modellbildung von Agentensystemen an. Sie bilden in [Burkhard 2003] die Grundlage eines ereignisorientierten Formalismus' zur Definition von Agentensystemen. Agenten und die Umwelt werden hier als Teilnetze modelliert, die ganz im Sinne Petris (siehe Seite 81) über Randstellen miteinander kommunizieren. Die Plätze entsprechen den inneren Zuständen der Agenten bzw. den äußeren Zuständen der Umwelt. Die Transitionen hingegen repräsentieren die Menge aller möglichen ausführbaren Aktionen bzw. eintretenden Umweltereignisse. Burkhard unterscheidet hierbei aus der Sicht eines Agenten drei wesentliche Gruppen von Transitionen (Aktionen/Ereignisse):

- ausführbare Aktionen
- beobachtbare Aktionen/Ereignisse
- weder ausführbare noch beobachtbare Aktionen/Eregnisse

Mit dieser Klassifikation kann nun der Erreichbarkeitsgraph eines ein Agentesystem beschreibenden Petrinetzes auf ein Transitionssystem reduziert werden, das die eingeschränkte Sicht eines Agenten formal beschreibt. Dies soll anhand des kurzen Beispiels erläutert werden.

Die Abbildung 4.4 zeigt das Petrinetz eines 2-Agenten-Systems. Beide Agenten können aus ihren Ruhezuständen (`s_a`, `s_b`) selbsttätig in die Folgezustände `w_a` und `w_b` wechseln, um dann dort konkurrierend auf die Zuteilung einer Ressource durch die Umwelt zu warten. Weiterhin sind in Abbildung 4.4 die für den Agenten `a` ausführbaren bzw. beobachtbaren Transitionen gekennzeichnet. Ausgehend von den ausführbaren und beobachtbaren Transitionen kann man nun ein reduziertes Transitionssystem erzeugen (Abdildung 4.5). Das Transitionssystem ähnelt einem nicht-determinierten Automaten

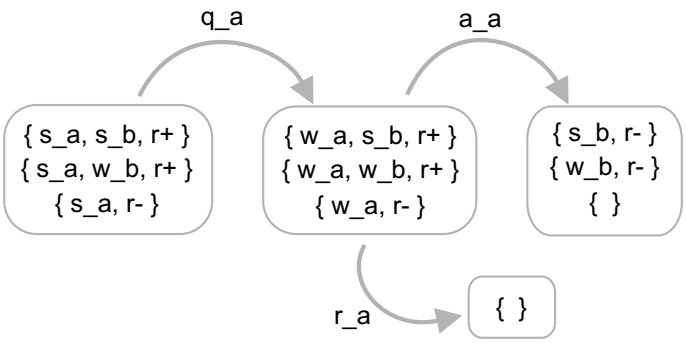


Abbildung 4.5: Reduziertes Transtitionssystem nach [Burkhard 2003]

und zeigt die möglichen Zustände, die vor bzw. nach den durch den Agenten `a` beobachtbaren bzw. ausführbaren Aktionen eintreten können.

4.2 Bondgraphen

Neben den informationsverarbeitenden Prozessen müssen bei automatisierungstechnischen Problemstellungen auch Phänomene der Energieumsetzung analysiert und untersucht werden. Mit den Bondgraphen hat Henry Paynter Ende der 50er des 20. Jahrhunderts eine Theorie entwickelt, die die domänenübergreifende Beschreibung von Energieumsetzungs- und -umwandlungsphänomenen ermöglicht [Paynter 1961]. Wie sich zeigen wird, handelt es sich bei den Bondgraphen um ein Beschreibungsmittel, mit dem ähnlich zu Petrinetzen Symmetrien zu materialen Produkteigenschaften hergestellt werden können (Strukturverträglichkeit).

4.2.1 Grundlagen

Wie bereits erwähnt zielen Bondgraphen darauf ab, Phänomene der Energieumsetzung wie z.B. Enerigeaustausch, -speicherung, -wandlung, -verzweigung und Dissipation innerhalb und zwischen physikalischen, materialen Systemen zu beschreiben. Diese Beschreibung beginnt (wie bei anderen Analysemethoden auch üblich) damit, innerhalb eines *universe of discourse U* das zu betrachtende System S zu indentifizieren und von der Umwelt (*environment*) E abzugrenzen. Die weitere Strukturierung besteht in der Dekomposition in Subsysteme und zuletzt in der Identifizierung von funktionellen Verbindungen zwischen den Subsystemen. Paynter nennt diesen Strukturierungsprozess *reticulation*:

> *The rational process of endowing a system with structure we call reticulation. The act of separating S from U, and thus defining the interface between S an E, is the first step in this process. The system is further reticulated by conceptually tearing it apart in to its essential elements. Since the structural attribute of a system which interests us most is the functional connectedness of its elements, the final step in the reticulation process is the sketching of the important relations and bonds of intersection among the elements and between each of the elements and the environment* [11]

4.2.1.1 Bonds

Die maßgeblichen funktionellen Verbindungen (*functional connectedness*) bei materialen Systemen sind diejenigen, die die Energieumsetzung ermöglichen. Diese Verbindungen werden in Bondgraphen durch die sogenannten Energie-*bonds* repräsentiert. Hierbei handelt es sich um energetische Verbindungen, die sich durch die folgenden Eigenschaften auszeichnen:

- Der Grad der Energieumsetzung, der über einem *bond* stattfindet, wird durch ein 2-Tupel von energetischen Größen beschrieben: *effort* and *flow*. Sie werden je nach

[11] [Paynter 1961], S. 11

Energieform so gewählt, dass ihr Produkt eine Leistung (zeitlicher Energiefluss) ergibt. Die folgende Tabelle zeigt die entsprechenden elektrischen, mechanischen und hydraulischen *flow*- und *effort*-Größen [KARNOPP 1975]. Auf eine ähnliche

	effort	*flow*
Mechanik translatorisch	Kraft (per)	Geschwindigkeit (trans)
Mechanik rotatorisch	Drehmoment (per)	Winkelgeschwindigkeit (trans)
Elektrik	Spannung (trans)	Strom (per)
Hydraulik	Druck (per)	Volumenstrom (trans)

Tabelle 4.1: *flow*- und *effort*-Größen in mechanischen, elektrischen und hydraulischen Systemen

Analogie stößt man, wenn man die physikalischen Leistungsvariablen nach der Anzahl der Messpunkte, die man zu deren Bestimmung benötigt, einordnet. Man unterscheidet die Leistungsvariablen dann in *per*-Größen (1-Punkt-Größen) und *trans*-Größen (2-Punkt-Größen) [SCHNIEDER 2002]. Bei dieser sogenannten Eulerschen Analogie entsprechen die meisten *per*-Größen den *effort*-Größen und die *trans*-Größen den *flow*-Größen, siehe Tabelle 4.1. Hier bildet jedoch die elektrische Spannung die Ausnahme, da diese *effort*-Größe als Potenzialdifferenz einer *trans*-Größe entspricht.

- *bonds* sind prinzipiell ungerichtet. Dadurch wird bewußt darauf verzichtet, Ursache-Wirkung-Zusammenhänge darzustellen. Die Bondgraphen gehören damit zu den nicht-kausalen Beschreibungsmitteln (siehe Kapitel 2.4.2.2). Wie in Abbildung 4.6 gezeigt, kann man einem *bond* zwei Paare von kausalen Signalkanten zuordnen. Im linken Fall prägt das Teilsystem S_1 dem Teilsystem S_2 die *effort*-Größe e auf, worauf S_2 mit dem *flow* f reagiert. Im rechten Bild ist die Kausalität umgekehrt.

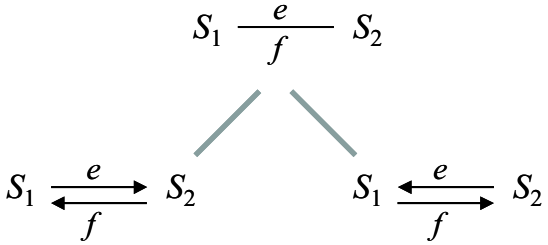


Abbildung 4.6: nicht-kausaler Energie-*bond* und mögliche äquivalente Signalkanten

Die eigentliche Modellierung mit Bondgraphen erfolgt ohne Berücksichtigung der Kausalitäten an den Energie-*bonds*. Diese werden erst in einem zweiten Analyseschritt ermittelt

und können dann im Bondgraphen mittels spezieller Kausalitätsmarkierungen (*causal stroke*) kenntlich gemacht werden. Die folgende Abbildung 4.7 zeigt zwei Energie-*bonds* mit unterschiedlicher Kausalität und den äquivalenten Signalflusskanten. Die Kausalität wird wie aus Abbildung 4.7 durch einen Querbalken am Ende des Energie-*bonds* dargestellt. Weiterhin besteht die Möglichkeit, gerichtete Energie-*bonds* in Bondgraphen

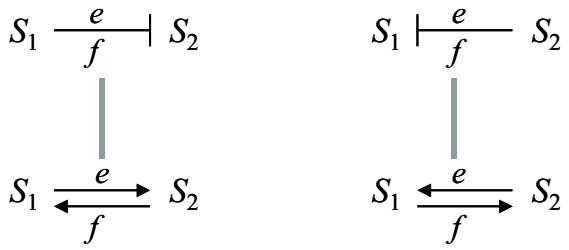


Abbildung 4.7: Energie-*bonds* mit kausaler Markierung (*causal stroke*)

zu verwenden. Die folgende Abbildung 4.8 zeigt einen Energie-*bond* mit Richtungssinn. Dieser Richtungssinn wird durch einen halben Pfeil kenntlich und dient allein dazu,

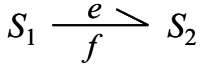


Abbildung 4.8: Energie-*bond* mit Richtungssinn

um die Richtung des Energieflusses darzustellen. Bei der Analyse von Bondgraphen bestimmt der Richtungssinn der Energie-*bonds*, welches Vorzeichen die entsprechenden *effort*- und *flow*-Größen in den resultierenden Bilanzgleichungen (z.B. Massenerhaltung, Kontinuitätsgleichung etc.) haben. Die prinzipielle Eigenschaft, dass Bondgraphen nichtkausal sind, bleibt davon unberührt.

4.2.1.2 Multiports

Wie im Zitat von Paynter auf Seite 86 deutlich wird, ist ein wichtiger Schritt des Retikulierens die Untergliederung eines Systems in seine essenziellen Bestandteile, mit denen die Phänomene der Energieumsetzung (Wandlung, Speicherung, Verteilung, Dissipation) beschrieben werden können. Paynter geht hier von der Idealisierung aus, dass die Energieumsetzungsphänomene auf diskrete, lokal konzentrierte Regionen - sogenannte *ports* - beschränkt sind:

> *... This amounts to assuming: (1) that the S-E-Interface is permeable to the passage of energy only over a relatively few areas of restricted extent - energy ports as we shall call them; (2) that energy storage function of the system is not distributed continuously throughout its volume, but is rather lumped in*

> *discrete localities or regions; (3) that the dissipative property of the system is also confined to descrete regions. Although this is admittedly an idealization, it is a very practical and necessary one.* [12]

Aufgrund dieser Abstraktion und unter Berücksichtigung der allgemeinen Leistungsvariablen in Tabelle 4.1 ist es nun möglich, einen beschränkten Satz grundlegender Energie-*ports* zu definieren, die für die unterschiedlichen physikalischen Disziplinen gleichermaßen gelten, und mit denen sich die wichtigsten energetischen Phänomene beschreiben lassen. Die folgende Tabelle 4.2 auf Seite 90 zeigt diese Energie-*ports*.

Wie in Tabelle 4.2 ersichtlich ist, haben die Energie-*ports* eine unterschiedliche Anzahl an Toren. Neben den energetischen Phänomenen, die durch die *ports* beschrieben werden, zeigt diese Abbildung auch ihre Symbole im Bondgraphen und in der letzen Spalte ihre Bestimmungsgleichungen.

Mit den 1-ports können energetische Quellen und Senken (z.B. Batterien, Strom-/Spannungsquellen), Energiespeicher (elektrischer Kondensator und Induktivität, Federn und Massen, Behälter und Rohre mit träger Strömung) und Dissipationselemente (Ohmscher Widerstand, Dämpfer, hydraulische Drosseln) beschrieben werden. Die Bestimmungsgleichungen von Widerständen sind i.A. nichtlineare *flow/effort*-Kennlinien. Das Speicherverhalten von Kapazitäten und Induktivitäten wird mit Integralen über der Zeit beschrieben.

Bei den 2-ports unterscheidet man Transformatoren und Gyratoren, die die Wandlung in unterschiedliche Energieformen ermöglichen. Ihre Bestimmungsgleichungen beschreiben Proporzionalitäten zwischen den Ein- und Ausgangsgrößen. Beispiele für Transformatoren sind z.B. Getriebe, Pumpen, hydraulische Zylinder und elektrische Transformatoren. Bei den elektromechanischen Wandlern (E-Motoren, Generatoren, Mikrophone usw.) handelt es sich oftmals um Gyratoren, da die dort wirkende Lorentz-Kraft die für Gyratoren typische Kreuzverkoppelung von Strom (*flow*) und Kraft(*effort*) hervorruft. Eine ähnliche Kreuzverkoppelung tritt beim Induktionsgesetz auf, bei dem die induzierte Spannung (*effort*) proporzional zur Drehzahl (*flow*) ist.

Wie die Tabelle 4.2 zeigt, bieten die Bondgraphen noch Verknüpfungselemente, mit denen *multiport*-Netzwerke aufgebaut werden können, in denen Energieverteilungsphänomene auftreten. Hier handelt es sich prinzipiell um 3-ports. Diese Elemente können jedoch leicht auf n-ports (multiports) mit einer beliebigen Anzahl von Toren generalisiert werden. Die Bestimmungsgleichen dieser multiports sind verallgemeinerte Kirchhoffsche Gesetze, so dass mit diesen Verknüpfungselementen Reihenschaltungen (Maschenregel) und Paralleschaltungen (Knotenregel) von Energie-*ports* dargestellt werden können. Reihen- und Parallelschaltungen sind nicht nur in elektrischen, sondern auch in mechanischen, hydraulischen und thermischen Systemen möglich.

Auf Seite 87 wurde gezeigt, dass es sich bei Bondgraphen um ein nicht-kausales Beschreibungsmittel handelt, bei dem ohne Berücksichtigung von Kausalitäten modelliert

[12] [Paynter 1961], S. 35

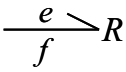
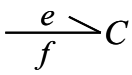
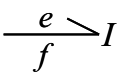
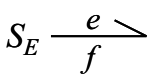
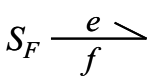
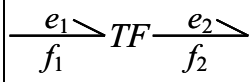
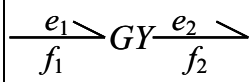
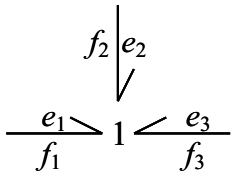
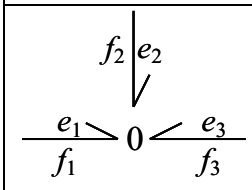
Energie-port	*Anzahl der Tore*	*energetisches Phänomen*	*Symbol im Bondgraphen*	*Bestimmungs-gleichung*
Widerstand	1	Dissipation		$e = \Phi_R(f)$
Kapaziät		Speicherung		$\int f dt = \Phi_C(e)$
Induktivität				$\int e dt = \Phi_I(f)$
effort-Quelle		Quelle, Senke		$e = e(t)$
flow-Quelle				$f = f(t)$
Transformator	2	Wandlung		$e_1 = m \cdot e_2$ $m \cdot f_1 = f_2$
Gyrator				$e_1 = n \cdot f_2$ $n \cdot f_1 = e_2$
Reihenschaltung	3 ... n	Verteilung		$e_1 + e_2 + e_3 = 0$ $f_1 = f_2 = f_3$
Parallelschaltung				$f_1 + f_2 + f_3 = 0$ $e_1 = e_2 = e_3$

Tabelle 4.2: Die wichtigsten Energie-*ports* zur Beschreibung energetischer Phänomene

wird. Jedoch implizieren die Bestimmungsgleichungen der Energie-*ports* bestimmte Kausalitäten, so dass damit die Kausalitäten der Energie-*bonds* nach erfolgter Modellierung in einem eigenen Analyseschritt berechnet werden können. Als Beispiel sei hier die Bestimmungsgleichung einer Kapazität erwähnt, die laut Tabelle 4.2 als Integral formuliert ist. Der Einfachheit halber werde die Kennlinie $\Phi_C(e)$ als Konstante C angenommen. Die Integralgleichung impliziert für die Kapazität (und für den an der Kapazität angeschlossenen Energie-*bond*) eine integrale Kausalität, siehe linke Seite in Abbildung 4.9

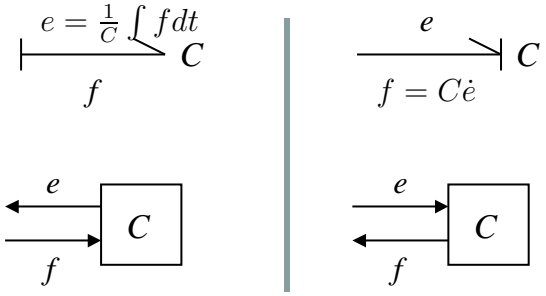


Abbildung 4.9: Kapazität mit integraler und derivativer Kausalität, $\Phi_C = C$

Anstatt als Integralgleichung lässt sich die Bestimmungsgleichung der Kapazität jedoch auch als Differenzialgleichung schreiben. Dementsprechend kehrt sich die Kausalität am Energie-*bond* um. Man spricht dann von derivativer Kausalität, Abbildung 4.9 rechte Seite. Um aus einem Bondgraphen ein Differenzial-Algebraisches Gleichungssystem erzeugen zu können, müssen für alle Speicherelemente derivative Kausalitäten gewählt werden. Ansonsten lassen sich nur gemischte Integral-Differenzial-Algebraische Gleichungen aufstellen, deren Lösung nur mit großem Aufwand möglich ist.

4.2.2 Schaltende Bondgraphen

Wie in Abschnitt 2.6.2 dargestellt, zeichnen sich hybride Prozesse durch die kooperative Wechselwirkung zwischen kontinuierlichen, funktionalen (=reversibel/determiniert) und diskreten, nicht-funktionalen Teilprozessen aus. Nach (2.95) erfordert diese kooperative Wechselwirkung die Abbildung zwischen kontinuierlichen und diskreten Trägermengen mittles Injektor/Quantisierer-Funktionen. Im Falle von Bondgraphen, die ja zur Beschreibung kontinuierlicher, funktionaler Prozesse dienen, bedeutet dies die Berücksichtigung diskreter Zustände mit Schalterelementen.

Es wird nun eine kurze tabellarische Übersicht über die unterschiedlichen Ansätze gegeben, mit denen schaltende Vorgänge in Bondgraphen modelliert werden können. Diese Modellierungsansätze werden dann miteinander verglichen. Zuvor jedoch werden kurz die möglichen Schaltvorgänge in physikalischen Systemen erläutert.

Bei den Schaltvorgängen kann man zwischen realen und idealen Schaltern unterscheiden. Reale physikalisch-energetische Schaltvorgänge laufen in makroskopischer Betrach-

tung ohne ersichtliche Sprünge ab, so dass die resultierenden Zustandstrajektorien *stetige*, nicht-differenzierbare Funktionen sind. Ideale Schaltvorgänge hingegen zeichnen sich durch sprunghafte Änderung der Zustandsgrößen aus. Die resultierenden Trajektorien verlaufen *nicht-stetig*. Desweiteren verletzen ideale Schaltvorgänge im Allgemeinen den Energieerhaltungssatz. Diese Vorgänge spiegeln zwar nicht das reale Verhalten wider, stellen aber bei sehr schnellen Ausgleichsvorgängen eine zulässige Abstraktion dar, denn die Simulation des realen Verhaltens bei schnellen Ausgleichsvorgängen ist oft nur mit sehr kleinen Schrittweiten möglich oder zieht andernfalls numerische Probleme und Instabilitäten nach sich [Borutzky 2000].

Die folgende Tabelle 4.3 zeigt nun einige Ansätze, um schaltende Vorgänge mit Bondgraphen zu modellieren. Aus der Tabelle geht hervor, welche Schaltarten von der jeweiligen Modellierungsmethode unterstützt werden, und ob sich die Kausalitäten während des Schaltvorganges verändern. Eine einfache Möglichkeit, schaltende Vorgänge in Bond-

Schaltmechanismus	*Quelle*	*Schaltart*	*Kausalität*
Kennlinien	[Beaman und Rosenberg 1988]	realer Schalter	fest
Schaltelement (switch)	[Strömberg 1994]	realer/idealer Schalter	variabel
schaltbare Kante	[Broenink und Wijbrans 1993]	idealer Schalter	fest
schaltbare *s*/*p*-Knoten	[Mostermann 1997]	idealer Schalter	variabel
modulierter Transformator	[Dauphin-Tanguy et al. 1989]	realer/idealer Schalter	fest

Tabelle 4.3: Auswahl einiger Ansätze für schaltende Bondgraphen

graphen darstellen besteht darin, Bondgraphenelemente wie Widerstände oder Speicher mit nichtlinearen Kennlinien zu versehen. Mit stetigen Kennlinien lassen sich somit reale Schaltvorgänge modelieren. In [Strömberg 1994] wird gezeigt, dass mit dieser Methode physikalische Sättigungserscheinungen wie z.B. Behälterüberlauf nur schwer modelliert werden können. Ein weiterer Nachteil ist die mangelnde Transparenz des Schaltvorgangs, da sich dieser hinter einer im Bondgraphen nicht sichtbaren Kennlinie verbirgt.

In [Strömberg 1994] wird ein neuer 1-port-Knoten (*switch*) eingeführt, der die Modellierung idealer Schaltvorgänge ermöglicht. Durch geschickte Verschaltung mit anderen 1-port-Knoten können mit diesem Schalterelement auch stetige Schaltvorgänge modelliert werden. Zu beachten ist, dass sich die Kausalität des Schalterelementes während der Simulationszeit verändern kann. Dies erschwert die Simulation, da sich dann auch die Struktur des zu lösenden DGL-Systems während der Simulation ändern kann.

In [Broenink 1993] wird vorgeschlagen, die Kanten der Bondgraphen als schaltbar zu definieren. Da über die Kanten die Energieverteilung stattfindet, können hiermit Ener-

gieflüsse unterbrochen werden, so dass ideale Schaltvorgänge modelliert werden können. Mostermann zeigt in [Mosterman 1997] anhand eines einfachen Modellierungsbeispiels, dass schaltende Kanten nicht einfach mit realen Schaltern verglichen werden können und es deshalb unter Umständen zu falschen Simulationsergebnissen kommt.

Ebendort ([Mosterman 1997]) wird das Konzept der schaltenden s/p-Verknüpfungen vorgestellt. Beim Abschalten einer Verknüpfung wird allen angeschlossenen Kanten ein Leistungsfluß von 0 aufgeprägt, so dass sich hier ähnlich wie bei [Strömberg 1994] ideale Schaltvorgänge ergeben, die die Energieerhaltung verletzen. Desweiteren können sich durch das Abschalten von Verknüpfungen zur Simulationszeit die Kausalitäten innerhalb des Bondgraphen ändern.

Das letzte hier vorgestellte Konzept zur Darstellung von Schaltvorgängen ist der modulierte Transformator nach [Dauphin-Tanguy 1989]. Schaltvorgänge werden dadurch berücksichtigt, dass der Übertragungsfaktor des Transformators je nach Schaltzustand 0 oder 1 ist. Die anliegenden *flow*/*effort*-Größen werden dann ebenfalls zu 0. Durch Kombination des Transformators mit Quellen- oder Widerstandsknoten können sowohl ideale als auch reale Schaltvorgänge modelliert werden. Zu beachten ist, dass es bei entsprechenden Kausalitäten am Transformator zu einer Division durch 0 kommen kann, so dass man solche Modellkonstrukte beim Bondgraphenentwurf vermeiden muss.

4.3 Produktmodell nach ISO 10303

Die Datenmengen, die bei Entwicklung, Fertigung und Betrieb industrieller Produkte anfallen, steigen sowohl mit der Komplexität der Produkte als auch mit dem Automatisierungsgrad in der Entwicklung und Produktion. Ein immer größerer Teil dieser Daten wird im Zuge der *CA*[13]-Technologien mit den unterschiedlichsten EDV-Anlagen gespeichert und liegt in verschiedenen Formaten und an verteilten Orten vor. Dadurch wird der Datenaustausch erschwert bzw. unmöglich gemacht. Der internationale Standard ISO 10303 mit seinen Normen soll es ermöglichen, alle Produktdaten, die während sämtlicher Produktkphasen benötigt werden, einheitlich darzustellen und rechentechnisch zu verarbeiten. Dies soll unabhängig von der Art des Produktes und des speichernden Systems gewährleistet werden. Der Standard trägt den Titel *Industrial automation systems and integration - Product data representation and exchange*. Eine andere Bezeichnung dieser Normen ist *Standard for the Exchange of product model data* und wird mit STEP abgekürzt. Die grundlegenden Ziele dieser Normenreihe gehen aus folgendem Zitat nochmals hervor:

> *This International Standard provides a representation of product information along with the necessary mechanisms and definitions to enable product data to be exchanged. The exchange is among different computer systems and environments associated with the complete product lifecycle, including product*

[13]CA steht für *computer aided*. Beispiele der *CA*-Technologien sind *CAD, CAM, CASE*

design, manufacture, use, maintenance, and final disposition of the product.[14]

Aufgrund ihrer Komplexität ist die ISO 10303 in unterschiedliche Reihen gegliedert, die auch Serien genannt werden. Dies sind im einzelnen

- Description methods (10er Serie)
- Implementation methods (20er Serie)
- Conformance testing methodology and framework (30er Serie)
- Integrated generic/application resources (40er und 100er Serie)
- Application protocols (200er Serie)
- Abstract test suites (300er Serie)
- Application interpreted constructs (500er Serie)
- Application Modules (1000er Serie)

Die Serien wiederum bestehen aus verschiedenen Teilen (*parts*). Diese Teile entsprechen einzelnen Normen. Sie sind durchnummeriert und werden durch Anhängen ihrer Ziffer an das Kürzel ISO 100303 gekennzeichnet. Beispiel: ISO 10303-33 entspricht dem *part* 33 aus der 30er Serie.

Die folgende Tabelle 4.4 zeigt die Serien von ISO 10303 und eine Auswahl aus den Teilen, die sie beinhalten.

Im folgenden wird geklärt, wozu die einzelnen Serien dienen und wie sie und ihre Teile miteinander in Beziehung stehen.

4.3.1 Aufbau und Inhalt von ISO 10303

ISO 10303 zielt ja darauf ab, produktrelevante Daten (Produktstruktur und Gestalt) plattformunabhängig speichern, austauschen und archivieren zu können. Hierbei wird ein Metamodellierungsansatz verfolgt, um die Repräsentation der Produktdaten von den Implementationsmethoden zum Datenaustausch zu trennen. Die Produktdaten werden in plattformunabhängigen Informationsmodellen, dem sogenannten integrierten Produktmodell, repräsentiert. Die eigentlichen Produktdaten sind dann Instanzen dieses Produktmodells. Die rechnerinterpretierbare Repräsentation der Instanzen wird in eigenen Normen spezifiziert.

[14] [ISO 1994a], S. 1

Serie	*Teil*	*Titel*
Description methods	1	Overview and fundamental principles
Teil 1-19	11	The EXPRESS language reference manual
	…	…
Implementation methods	21	Clear text encoding of the exchange structure
Teil 20-29	25	EXPRESS to OMG XMI binding
	28	XML representation to EXPRESS schemas and data
	…	…
Conformance testing and methodology framework	31	General concepts
	34	Abstract test methods
Teil 30-39	…	…
Integrated generic resources	41	Fundamentals of product description and support
	42	Geometric and topological representation
Teil 40-49	44	Product structure configuration
	…	…
Integrated application resources	101	Draughting
	102	Ship structures
Teil 100-199	104	Finite Element analysis
	…	…
Application protocols	201	Explicit draughting
Teil 200-299	212	Electrotechnical design and installation
	214	Core data for automotive mechanical design processes
	221	Functional data and their schematic representation for process plant
	...	…
Abstract test suites	301	Explicit draughting
Teil 300-399	212	Electrotechnical design and installation
	214	Core data for automotive mechanical design processes
	…	…
Application interpreted constructs	501	Edge-based wireframe
	502	Shell-based wireframe
Teil 500-599	…	…
Application Modules	1001	Appearance assignment
Teil 1000 - …	1002	Colour
	…	…

Tabelle 4.4: Überblick über die STEP-Normen

Beschreibungsmittel und Implementationsmethoden

Das Beschreibungsmittel, das zur Darstellung der Informationsmodelle dient, heißt EXPRESS und wird in der 10er-Serie von ISO 10303 definiert ([ISO 1994b]). EXPRESS ist eine Ontologiespezifikationssprache und stellt elementare Typen wie z.B. *integer* und *string* zur Verfügung. Weiterhin ist die Definition eigener Datentypen, sogenannter *entities*, möglich, die mit den Klassen objektorientierter Sprachen verglichen werden können. Der *entity*-Datentyp unterstützt Konzeptualisierung und Taxonomien (siehe Abschnitt 3.1.2).

Die Implementationsmethoden werden in der 20er-Serie beschrieben. Sie definieren sprachspezifische Mechanismen zur Verarbeitung der mit EXPRESS formalisierten Produktdaten. Dies geschieht, indem die EXPRESS-Konstrukte, aus denen die Informationsmodelle aufgebaut sind, auf die Sprache abgebildet werden, die den Methoden zugrunde liegt. Da EXPRESS vielfältige Ausdrucksmöglichkeiten bietet, ist die Abbildung im Allgemeinen nicht vollständig (nicht alle EXPRESS-Konstrukte finden einen Gegenpart, es handelt sich im streng mathematischen Sinne also auch nicht um eine Abbildung). Der Teil 21 der Implementationsmethoden beschreibt eine Austauschstruktur, mit der Pro-

duktdaten in einer Datei abgelegt werden können. Teil 22 beschreibt das *Standard Data Access Interface*, das den Zugriff auf die Produktdaten in Datenbanken ermöglicht. Einige andere Teile spezifizieren, wie das SDAI mit konventionellen Programmiersprachen (z.B. C, C++, Java) implementiert wird.

Die Abbildung der Datenmodellierungssprache EXPRESS auf XMI wird im Teil 25 der STEP-Normenreihe ISO 10303 spezifiziert [ISO 2001]. Ziel dieser Teilnorm ist die Erzeugung von XML-Schemas aus EXPRESS-Datenmodellen, um diese mit UML-Werkzeugen wiederverwenden zu können. Die Abbildung der grafischen Sprachelemente von EXPRESS-G auf UML Klassendiagramme ist nicht explizit formuliert, sondern läßt sich nur aus den entsprechenden XMI-Fragmenten herleiten. Da EXPRESS über mehr Sprachkonstrukte zur Datenmodellierung als die Klassendiagramme in UML vefügt, ist die Abbildung von EXPRESS auf XMI nur unvollständig. Deshalb fehlen die EXPRESS-typischen Regel-Deklarationen, Funktionen, Prozeduren, Konstanten und Vererbungsmechanismen für komplexe Instanzen.

Der Teil 28 (siehe [ISO 2002]) verfolgt ein ähnliches Ziel, geht aber nicht den Umweg über XMI, sondern ermöglicht es, EXPRESS-Datenmodelle und EXPRESS-Instanzen ohne ein XMI-konformes Schema direkt als XML-Dateien zu speichern. Nach [Lubell 2002] werden das *late binding* und das *early binding* unterschieden. Beim *late binding* gibt es ein vom Datenmodell unabhängiges XML-Schema, nachdem sich alle XML-Modelldateien richten. Beim *early binding* hingegen muss ähnlich zu XMI zuerst ein vom Datenmodell abhängiges XML-Schema erzeugt werden. Die zusätzliche Schema-Erzeugung hat den Vorteil, dass die XML-Dateien übersichtlicher und implementierungsfreundlicher werden.

Produktmodell

Das eigentliche Produktmodell von ISO 10303 wird in den 40er, 100er und 200er Serien definiert und ist in unterschiedliche semantische Ebenen unterteilt. Die integrierten Resourcen der 40er Serie (*Integrated generic resources* und 100er Serie *Integrated application resources*) bilden den branchen- und anwendungsunabhängigen Teil des Produktmodells und beschreiben grundlegende Produkteigenschaften wie z.B. Gestalt, Material, Produktstruktur und -aufbau, Toleranzen, Versionierung, Freigabe usw. In den Anwendungsprotokollen der 200er Serie (*application protocol*) werden die branchenspezifischen Produktinformationen modelliert. Sie bilden aus ontologischer Sicht den zentralen Bestandteil von ISO 10303. Die Anwendungsprotokolle sind in facheigener Terminologie formuliert und sind somit nach [Ortner 2000b] fachsprachliche Schemen. So gibt es Anwendungsprotokolle für die Automobilindustrie, die Elektroindustrie, den Schiffbau, verfahrenstechnische Anlagen usw., siehe Tabelle 4.4.

Den wichtigsten Bestandteil der Anwendungsprotokolle stellt wiederum das ARM (*Application Reference Model*) dar. Bei dem ARM handelt es sich um das eigentliche anwendungsspezifische Produktmodell mit facheigener Terminologie. Es wird mit EXPRESS und analog dazu mit EXPRESS-G formuliert und ist in *Units of Functionality*

(UoF) unterteilt. Innerhalb der UoF werden *entities* definiert, die durch Vererbung und Attributbildung in festen Beziehungen zueinander stehen und in diesem Zusammenhang etwas irritierend[15] Anwendungsobjekte genannt werden (*application object*). Ein weiterer Bestandteil der Anwendungsprotokolle ist das AIM (*Application Interpreted Modell*), das dazu dient, das ARM auf die intergrierten Resourcen abzubilden. In einer Abbildungstabelle (*mapping table*) wird jedem Anwendungsobjekt einer UoF und seinen Attributen ein AIM-Element zugeordent, das direkt mit einer *entity* aus den integrierten Resourcen korrenspondiert oder sich von ihr durch Vererbung ableiten lassen. Somit ist gewährleistet, dass sich die Produktdaten eines Anwendungsprotokolls als Instanzen der integrierten Resourcen darstellen lassen können.

Mit der Aufteilung des Produktmodells in unterschiedliche semantische Ebenen folgt ISO 10303 dem Konzept zur Ontologieintegration nach [Guarino 1995b]: die integrierten Ressourcen entsprechen domänenspezifischen Ontologien. Sie sind zwar branchenneutral formuliert, definieren aber für die Domäne 'Produktdatentechnologie' spezifische Konzepte. Die *ARM* der Anwendungsprotokolle hingegen entsprechen Anwendungsontologien (*application ontologies*). Die AIM vermitteln zwischen diesen Ontologien, indem sie die Konzepte der Anwendungsontologien auf die domänenspezifischen Ontologien abbilden.

Testmethoden und -fälle

ISO 10303 bietet mit der 30er Serie (*Conformance testing methodology and framework*) und der 300er Serie (*Abstract test suite*) die Testfälle und -methoden, um die Übereinstimmung einer Implementation mit einem Anwendungsprotokoll zu überprüfen. Während die 30er Serie die Testmethoden festgelegt, stehen in der 300er Serie die zu jedem Anwendungsprotokoll passenden Testfälle. Die gesamten Anforderungen an eine Implementation werden in den *Conformance requirements* des Anwendungsprotokolls zusammengefasst. Beispielsweise wird im AP 201 gefordert, dass eine Implementation dieses Protokolls die Implementationsmethode ISO 10303-21 (*Clear text encoding of the exchange structure*) unterstützt und den Testmethoden aus ISO10303-34 in allen Testfällen aus ISO 10303-301 genügt.

Anwendungskonstrukte und -module

Die 500er und 1000er Serien enthalten die sogenannten Anwendungskonstrukte und die Anwendungsmodule. Es handelt sich hierbei um eine Sammlung von Informationsmodellen, die AP-übergreifende, wiederverwendbare Konzepte beschreiben und den STEP-Entwicklern als Entwurfsmuster für neue Anwendungsprotokolle dienen. Da diesen Informationsmodellen im Gegensatz zu den Anwendunsprotokollen abstraktere Gegenstandsbereiche zugrunde liegen (z.B. Farbe, Kurven, Datum und Zeit), können sie nach Guarino als *top-level*-Ontologien bezeichnet werden.

[15] *entities* entsprechen nicht Objekten, sonder vielmehr Klassen

4.3.2 Anwendungsprotokolle für die Automatisierungstechnik

Wie bereits auf Seite 96 erläutert wurde, stellen aus sowohl ontologischer als auch rein pragmatischer Sicht die Anwendungsprotokolle (200er Serie) den zentralen Bestandteil der ISO 10303 Normenreihe dar, da sie mit den ARM den fachsprachlichen Teil und damit die eigentichen Anwendungsontologien des STEP-Produktmodells definieren. Die anderen Normenreihen stellen lediglich Dienste zur Entwicklung, Implementierung und Nutzung der Anwendungsprotokolle zur Verfügung: Beschreibungsmittel (10er Serie), Implementationsmethoden (20er Serie), Testmethoden/Testfälle (30er und 300er Serien), domänenspezifische und *top-level*-Ontologien (40er, 100er, 500er und 1000er Serien). Der Einsatz des STEP-Produktmodells für die Automatisierungstechnik beginnt also mit der Wahl eines passenden Anwendungsprotokolls. Da bisher noch kein Anwendungsprotokoll existiert, das auf die Anforderungen der Automatisierungstechnik zugeschnitten ist, müssen die bereits bestehenden Anwendungsprotokolle auf ihre Tauglichkeit hin untersucht werden. Nach einer ersten Grobauswahl kommen nur zwei Anwendungsprotokoll in die engeren Wahl:

- **AP 212** *Electrotechnical design and installation*

- **AP 221** *Functional data and their schematic representation for process plant*

Beide werden im folgende kurz vorgestellt und miteinander verglichen. Aufgrund der Komplexität[16] solcher Anwendungsprotokolle bietet sich die Vorstellung anhand der *data planing models* an. In ihnen werden werden die UoF eines Anwendungsprotokolls und die Relationen zwischen den UoF grafisch und übersichtlich dargestellt.

4.3.2.1 AP212: Electrotechnical Design and installation

Der Zweck dieses Anwendungsprotokolls ist der Austausch von Produktinformationen für elektrotechnische Anlagen:

> *This document establishes the information requirements for the exchange of product information of electrotechnical systems, using the data needed for specifying the product in the design process. It defines as an application protocol the implementation of such exchanges using the STEP integrated resources.*[17]

Das *data planing model* in Abbildung 4.10 zeigt die wichtigsten UoF von AP 212 und deren Beziehungen untereinander [Däubler 1998]. Zunächst kann man zwischen UoF mit technischem Inhalt (Anwendungsobjekte für Geräte, Funktionen usw.; grau hinterlegt) und mit administrativem Inhalt (Andwendungsobjekte für Freigabe, Dokumentation usw.; weiß hinterlegt) unterscheiden.

[16] AP 212 und AP 221 umfassen beide mehrere hundert Anwendungsobjekte, siehe S. 96
[17] [ISO 1996], Deckblatt

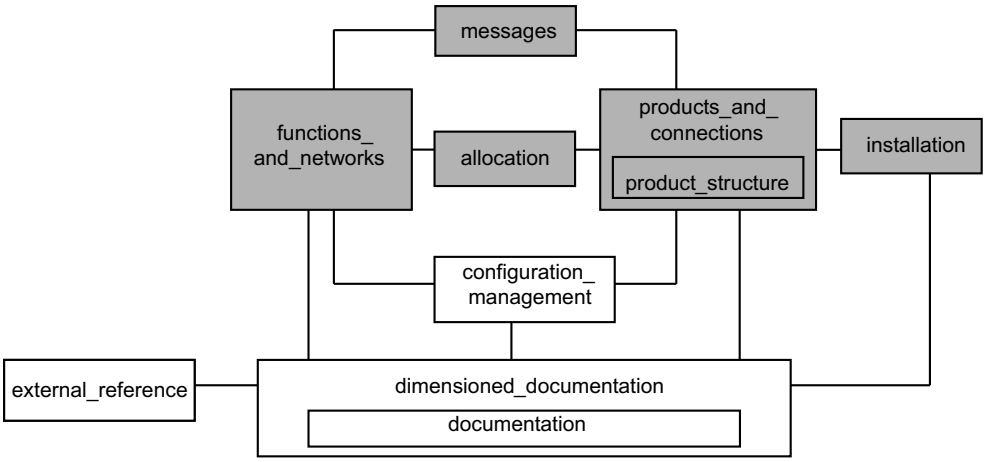


Abbildung 4.10: *data planing model* von AP 212 mit den wichtigsten *Units of Functionality* (UoF)

Die technische UoF functions_and_networks definiert das funktionsorientierte Teilmodell, products_and_connections und product_structure definieren das produktorientierte Teilmodell von AP 212. Diese beiden zentralen Teilmodelle werden durch spezielle Anwendungsobjekte aus der UoF allocation einander zugeordnet. Damit kann ein Zusammenhang zwischen Geräten und den von ihnen ausgeführten Funktionen hergestellt werden. Neben dem funktions- und produktorientierten Teilmodell gibt es ein ortsorientiertes Teilmodell, das durch die UoF installation definiert wird. Damit kann sowohl die genaue Anordnung von Produkten als die Verlegung von Leitungen und Verbindungskabeln beschrieben werden. Der Informationsfluss, der innerhalb eines elektrotechnischen Systems erfolgt, wird durch Anwendungsobjekte der UoF messages spezifiziert. Nachrichten können sowohl im funktions- also auch im produktorientierten Teilmodell ausgetauscht werden.

Die administrativen Teilmodelle enthalten als wichtigste UoF configuration_management für Produktvarianten und Freigaben, documentation und dimensioned_documentation für grafisch/textuelle Produktbeschreibungen und external_reference zum Einbinden externe Produktbeschreibungen, deren Inhalt ausserhalb von AP 212 liegen wie z.B. Dokumente, physikalische Modelle, Simulationsdaten usw.

4.3.2.2 AP221: Functional data and their schematic representation for process plant

Das AP 221 dient, wie bereits aus dem Titel hervorgeht, dem Austausch von Produktinformationen für verfahrenstechnische Anlagen:

> *This document specifies the information model for part 221 (...) of ISO 10303. It addresses the objects (systems and equipment) within a process*

> *plant, their idenification, classification, conncectivity, composition and properties. It also addresses representation of the objects as a piping an instrumentation diagram (P&ID).*[18]

Ein Ausschnitt aus dem data planing model von AP 221 ist in Abbildung 4.11 dargestellt. Ähnlich wie beim AP 212 besteht auch das AP 221 aus unterschiedlichen UoF mit technischem oder administrativem Informationsgehalt (engineering content und business practice). Die UoF mit technischem Informationsgehalt sind in Abbildung 4.11 grau hinterlegt.

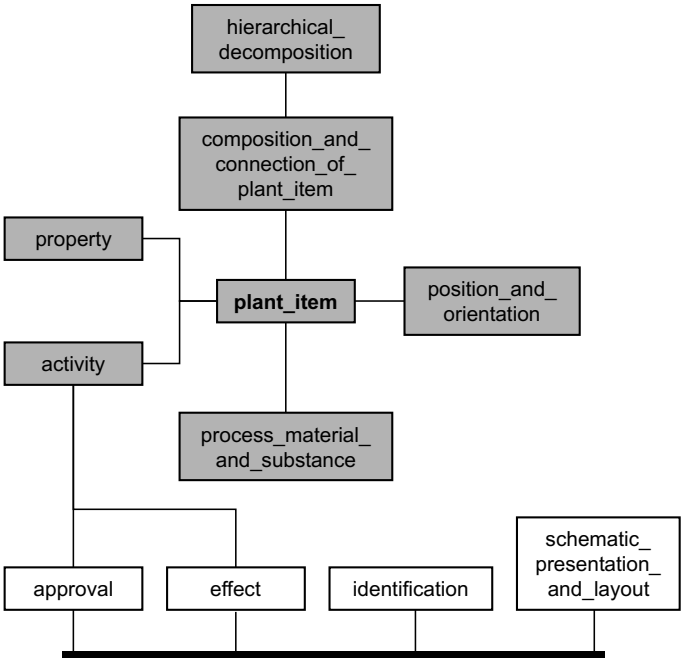


Abbildung 4.11: *data planing model* von AP221 mit den wichtigsten *Units of Functionality* (UoF)

Die zentrale UoF von AP 221 ist plant_item. Sie enthält die Anwendungsobjekte material und facility, die die physikalische/materiale und funktionale Sicht auf Anlagen widerspiegeln. Alle mit diesen beiden dualen Sichten verbundenen Konzepte wie Hierarchie, Connectivity/Vernetzung, Lage, Identifikation/Asset-Management werden in anderen UoF modelliert: composition_and_connection_of_plant_item beschreibt Aufbau und Vernetzung von funktionalen/materialen Anlagekomponenten. Die eng damit verbundene UoF hierarchical_decomposition ist ein Teilmodell für Baugruppen und Anlagenhierarchien. In der UoF position_and_orientation werden topologische Anlageninformationen (Montageort und -lage, z.B. Halle, Raum, Ebene) beschrieben. Genaue Ortskoordinaten können nur mit dem AP 227 *Plant Spatial Configuration* ausgetauscht werden. Die für verfahrenstechnische Anlagen notwendigen Stoffinformationen sind in

[18] [ISO 1997], Deckblatt

process_material_and_substance beschrieben. Hier wird zwischen Stoffen von Produkten/Edukten und Materialen von verarbeitenden Anlagenteilen/Ressourcen unterschieden. Die UoF activity stellt das Teilmodell für alle mit verfahrenstechnischen Anlagen verbundenen Prozesse und Aktivitäten dar. Hierunter fallen sämtliche Produktionsprozesse und auch alle Entwicklungsaktivitäten beim Anlagen-Design.

Neben dem oben beschriebenen *engineering content* enthält das Produktmodell von AP 221 eine Vielzahl administrativer UoF, die Teilmodelle für die Anlagenidentifikation beim *asset management*, Freigabe, Versionierung, Konfigurationsmanagement usw. enthalten und hier nicht weiter beschrieben werden sollen.

4.3.2.3 Vergleich und Auswahl

Nach der kurzen Vorstellung der beiden Anwendungsprotokolle wird nun im folgenden ihre Tauglichkeit zur Beschreibung von Gerätestrukturen automatisierungstechnischer Systeme erörtert.

Zunächst fällt auf, dass beide Protokolle ein hierarchisches Produktmodell unterstützen und mit ihren Teilmodellen in eine funktionale und eine materiale Anlagensicht unterscheiden. Das AP 221 hat darüber hinaus den Vorteil, dass die Beschreibung von Produkten/Edukten, von verarbeitenden Prozessressourcen und von Objetkprozessen möglich ist. Nachteilig erweist sich beim AP 221, dass sich die beiden zentralen Anwendungsobjekte zur Beschreibung funktionaler und materialer Anlagensichten (facility und material) in derselben UoF plant_item befinden. Hier erfolgt im AP 212 eine klare Trennung von materialer und funktionaler Sicht in einzelne UoF mit funktionsorientiertem und produktorientiertem Teilmodell (functions_and_networks und product_structure). Weiterhin enthalten diese getrennten Teilmodelle eigene Hierarchie-Dienste, die im AP 221 in einer eigenen UoF ausgelagert sind.

Zusammenfassend lässt sich feststellen, dass das AP 221 zwar das aus automatisierungstechnischer Sicht umfassendere Produktmodell ist. Das produktorientierte Teilmodell von AP 212 ist jedoch besser zur Darstellung von Gerätestrukturen und Netzwerken ausgestattet und leichter nutzbar. Insofern wird das AP 212 *Eletrical design and installation* als produktbeschreibende Ontologie in dieser Arbeit eingesetzt.

4.3.3 Das produktorientierte Teilmodell von AP 212

Die Abbildung 4.12 auf Seite 107 zeigt nun das produktorientierte Teilmodell von AP 212 als EXPRESS-G Datenmodell. Wie in Abbildung 4.10 gezeigt setzt sich dieses Teilmodell aus der UoF product_structure, deren *entities* sich im linken bzw. unteren Teil der Abbildung befinden, und aus der UoF products_and_connections im rechten bzw. oberen Teil zusammen.

Wichtigstes Element des Diagramms ist die Entität device, die elektrotechnische Geräte im weitesten Sinne repräsentiert. Geräte können mit der Entität assembly_relationship

zu Baugruppen angeordnet werden. Über die Elemente design_discipline_item_definition und item_version sind die Geräte mit den Versionsverwaltungselementen des Produktmodells verbunden. Eine weitere wichtige Entität stellt connection dar, die von connectivity_definition erbt. Instanzen dieser Entität repräsentieren die physikalischen Verbindungen, die zwischen elektrotechnischen Geräten z.B. in Form von Leitungen und Kabeln gelegt werden können. Dabei verbinden die connection-Instanzen die Geräte über terminal-Instanzen, die über das terminal_of-Attribut auf die entsprechenden Geräte verweisen. Desweiteren können mit der Entität connection_bundel auch Bündel von Verbindungen spezifziert werden.

Die restlichen Entitäten sollen hier nicht näher untersucht werden, da sie im Verlaufe der Arbeit nicht verwendet werden. Desweiteren wird eine reduzierte Form dieses produktorientierten Teilmodells in Kapitel 5.1.3 auf Seite 112 vorgestellt und im Detail erläutert.

4.4 Formale Prozessbeschreibung nach VDI/VDE 3682

Gegenstand der Verfahrenstechnik ist im weitesten Sinne die Herstellung chemischer Erzeugnisse. Dies geschieht in Fließ- oder *batch*-Prozessen[19], die sich von fertigungstechnischen Stückprozessen durch kontinuierliche, räumliche oder zeitliche Zustandsänderungen unterscheiden. Die Führung (Koordination, Steuerung, Regelung) bzw. Leitung solcher verfahrenstechnischer Prozesse wird mit prozessleittechnischen Anlagen durchgeführt (siehe [Polke 1994], S. 13), wobei der Anteil an reiner Automatisierungstechnik im Sinne einer direkten, geschlossenen Rechner-Prozess-Kopplung (*online*, siehe [Schnieder 1993], S. 43) einen eher kleinen Teil innerhalb eines Prozessleitsystems ausmacht.

Trotzallem bleibt die ingenieurmäßige, verständliche, nachvollziehbare und während des Lebenszyklus' der Anlage gültige Beschreibung von verfahrenstechnischen Prozessen eine große Herausforderung, der sich die Prozessleittechnik stellen muss. Die besondere Anforderung liegt dabei darin, sowohl den Prozess als auch die bei Planung, Einrichtung und Betrieb einer Anlage anfallenden Informationen (Anlagendokumentation) kombiniert beschreiben zu können. Die von der GMA[20] entwickelte VDI/VDE-Richtlinie 3682 [GMA 2003] stellt den aktuellsten Ansatz dar, die Prozesssicht mit den zur Anlagendokumentation notwendigen Informationsstrukturen zu integrieren.

Die Richtlinie wurde von einem Expertengremium hinsichtlich Verständlichkeit, hohem Formalisierungsgrad und Dekomponierbarkeit/Erweiterbarkeit konzipiert. Sie umfasst zunächst ein grafisches, formales Beschreibungsmittel für verfahrenstechnische Prozesse aus funktionaler Sicht. Diese Prozessbeschreibung basiert zwar auf einem aus dem

[19] Nach [Pallaske 1994] handelt es sich bei allen in Durchfluss-Apparaten durchgeführten Verfahren um raum- und zeitveränderliche instätionäre Fließprozesse mit den Spezialfällen *Kontiprozess* (nur räumlich variabel, zeitlich konstant) und *batch-Prozess* (nur zeitliche variabel, räumlich konstant).

[20] VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik, Fachausschuss 7.21 'Formalisierte Prozessbeschreibung'

Software-Engineering entlehnten Phasenmodell (siehe [Lauber 1996], S. 22) und wird deshalb auch *Phasenmodell der Produktion* genannt, weist aber in seiner aktuellen Ausprägung viele Gemeinsamkeiten mit Kanal/Instanzen-Nezten auf. Auf die Gemeinsamkeiten wird im Folgenden kurz eingegangen. Der Hauptunterschied zwischen beiden Beschreibungsmitteln liegt laut [Lauber 1996], S. 26 aber darin, dass im Phasenmodell kontinuierliche und in Petrinetzen diskrete Zustandsübergänge stattfinden. In mehreren Arbeiten konnte jedoch gezeigt werden, dass sich mit Petrinetze auch kontinuierliche Vorgänge modellieren lassen, siehe [Chouikha 1999] und [Decknatel 2001].

Ähnlich wie bei Kanal/Instanzen-Netzen unterscheidet man bei der formalen Prozessbeschreibung passive Prozessobjekte (Produkte, Energien) und aktive Prozessobjekte (Prozessoperatoren). Die Operatoren transformieren die Produkte/Energien von einem Vorzustand in einen Nachzustand. Der Fluss von Produkten/Energien wird durch gerichtete Kanten dargestellt, wobei zwischen den unterschiedlichen Zuständen eines passiven Prozessobjektes immer ein Prozessoperator geschaltet sein muss. Die folgende Abbildung 4.13 zeigt, welche grafischen Symbole in der VDI/VDE-Richtlinie Verwendung finden.

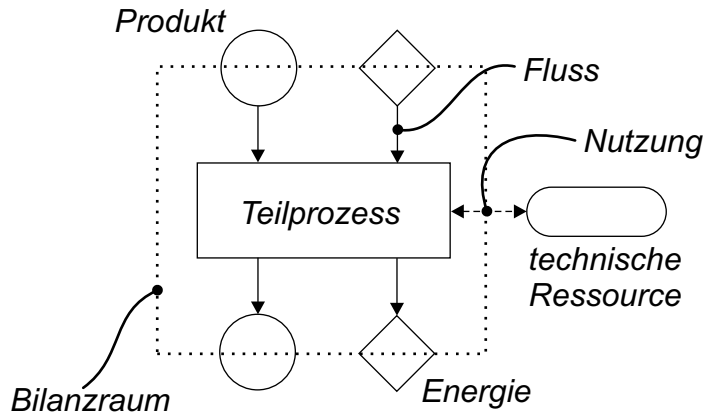


Abbildung 4.13: Grafische Elemente der formalen Prozessbeschreibung [GMA 2003]

Prozessoperatoren werden durch Rechtecke dargestellt, Produkte durch Kreise, Energien durch Rauten und technische Ressourcen durch abgerundete Rechtecke. Die Kanten ermöglichen die Darstellung von Produkt/Energieflüssen und die Darstellung von Nutzungsrelationen zwischen Prozessoperatoren und Ressourcen. Ein für die Verfahrenstechnik wichtiges Hilfsmittel ist die Darstellung von Bilanzräumen durch gestrichelte Rahmen. Diese ermöglichen die Bilanzierung von Prozessen über deren Bilanzgrenzen hinweg. Um die Struktur komplexer Prozesse darstellen zu können, kann man Prozessoperatoren durch Dekomposition verfeinern. Die Regeln für diese Verfeinerung von Prozessoperatoren entsprechen im Wesentlichen den Regeln für die Transitionsverfeinerung in Petrinetzen (vgl. hierzug [Reisig 1985]).

Neben dem graphischen Beschreibungsmittel enthält die Richtlinie ein UML-basiertes Informationsmodell, das die Syntax und Semantik der Prozessbeschreibung definiert und die Attributierung aller in der Prozessbeschreibung verwendeten Prozessobjekte ermög-

licht. So sind die notwendigen Datenstrukturen geschaffen, um die bei der Anlagendokumentation anfallenden Informationen beschreiben zu können.

Das folgende Klassendiagramm zeigt einen Auschnitt aus diesem Informationsmodell. Wie in dem Klassendiagramm zu sehen ist, besteht ein Prozess analog zur grafischen Prozessbeschreibung aus Prozessoperatoren und aus passiven Prozessobjekten, die durch unterschiedliche Relationen (Nutzung, Bilanzgrenze, Fluss) einander zugeordnet sind. Die passiven Prozessobjekte wiederum untergliedern sich in Energien oder Produkte. Diese werden durch die Prozessoperatoren einer Transformation unterzogen, wobei die Operatoren hierzu eine technische Ressource nutzen. Darüberhinaus kann ein Operator durch einen Bilanzraum begrenzt sein.

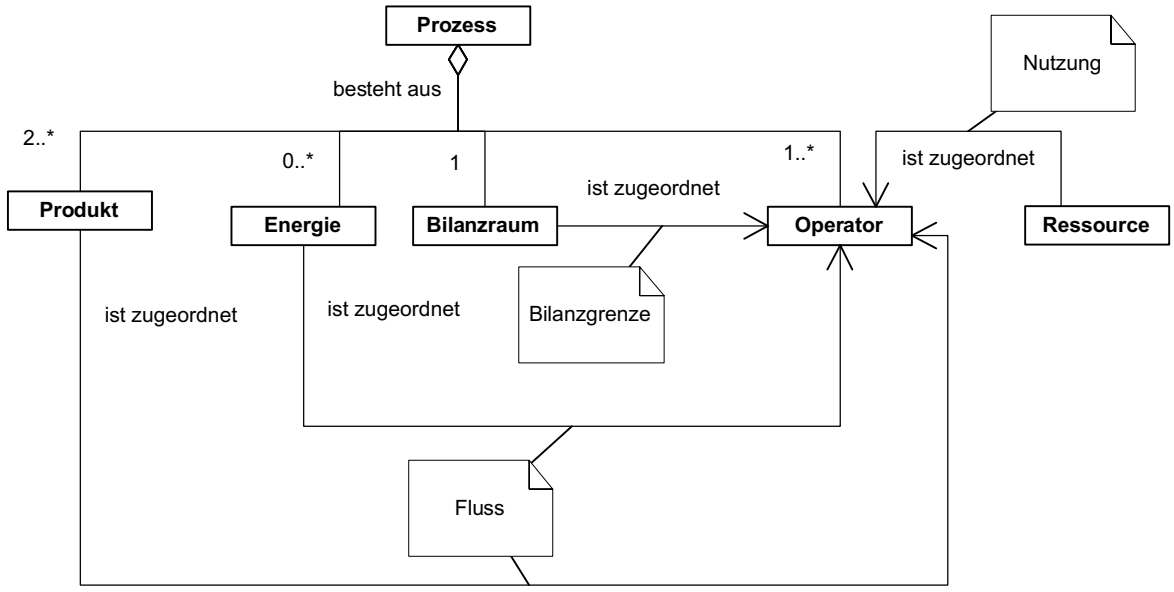


Abbildung 4.14: Ausschnitt aus dem Informationsmodell der formalen Prozessbeschreibung [GMA 2003]

Jeder Prozess und alle Prozessobjekt können mit einem Attribut zur Kennzeichnung (Ident, Version, Name usw.) und mit beliebig vielen Merkmalsattributen (Kategorie, beschreibender und beziehungsherstellender Bestandteil) versehen werden. Diese Attributierung ermöglicht dann letztendlich den Informationsaustausch mit CAE-Systemen, die bei Errichtung und Betrieb verfahrenstechnischer Anlagen verwendet werden. Denkbar wäre z.B. der Export einer Messstellenliste in ein Leitsystem oder der Import von Informationen zur Qualitätssicherung.

Die formalisierte Prozessbeschreibung nach VDI/VDE-Richtlinie 3682 stellt einen richtungsweisenden Ansatz dar, die funktionale Prozesssicht mit den für die Anlagendokumentation notwendigen Informationsstrukturen zu koppeln und ebnet damit den Weg für einen integrierten Entwurfsprozess, wie er in der Einleitung dieser Arbeit bereits skizziert wurde. Das in den folgenden Kapiteln erläuterte Verfahren zur strukturtreuen Modellbildung mit strukturverträglichen Ontologien würde sich daher hervorragend

eignen, funktionale Prozessmodelle auf der Basis bestehender Anlagendokumentation konsistent und widerspruchsfrei aufzubauen. Voraussetzung hierfür wäre, dass notwendige Teile der Anlagendokumentation als rechnerinterpretierbare Datenmodelle vorliegen müsste. Dieser Anwendungsfall konnte im Rahmen der Dissertation nicht berücksichtigt werden, birgt jeoch viel Potenzial für nachfolgende Studien.

4.5 Zusammenfassung

Da die Automatisierungstechnik als ausgesprochen interdisziplinäre Wissenschaft verstanden werden muss, kommen bei der Modellbildung automatisierungstechnischer Prozesse und Systeme viele unterschiedliche Beschreibungemittel zum Einsatz. Neben den reinen Beschreibungsmittlen für die Verhaltensspezifikation werden auch spezielle Informationsmodelle genutzt, mit denen die funktionalen oder strukturellen Eigenschaften automatisierungstechnischer Systeme und Einrichtungen beschrieben werden können.

In diesem Kapitel wurden beispielhaft Bondgraphen, Petrinetze, das Produktmodell nach ISO 10303 und die formalisierte Prozessbeschreibung nach VDI/VDE 3682 vorgestellt. Petrinetze und Bondgraphen ermöglichen die Verhaltensbeschreibung informatischer und energetischer Prozesse, beinhalten aber auch eine Struktursicht, die mit Gerätebeschreibungen nach ISO 10303 korreliert. Basierend auf diesen Beschreibungsmitteln werden im weiteren Verlauf der Arbeit strukturverträgliche Ontologien formuliert und deren Strukturverträglichkeit definiert und rechnerunterstützt überprüft.

Petrinetze wurden von Carl Adam Petri für die Modellierung informationsverarbeitender Maschinen zur Lösung rekursiver Aufgaben entwickelt, eignen sich aber ganz allgemein zur Darstellung beliebiger organisatorischer Vorgänge. Insbesondere ihre Einsatzmöglichkeit in der Automatisierungstechnik wurde anhand einiger Beispiele (Steuerungstechnik, Verfahrenstechnik, Agentensysteme) erläutert.

Mit den von Henry Paynter entwickelten Bondgraphen können Energieumsetzungs- und umwandlungsphänomene domänenübergreifend dargestellt werden. Sie ermöglichen es, technische Systeme mit einem methodischen Strukturierungsprozess (*reticulation*) zielgerichtet zu analysieren. Neben den Grundlagen der Bondgraphen wurde in diesem Kapitel auch auf die unterschiedlichen Ansätze, Schaltphänomene mit Bondgraphen darzustellen, eingegangen.

Im Gegensatz zu den bereits genannten Beschreibungsmitteln dient das Produktmodell von ISO 10303 zur Beschreibung statischer Systemeigenschaften wie Gestalt, Aufbau, Funktionsgruppen usw. Aus der Vielzahl der in ISO 10303 enthaltenen Anwendungsprotokolle wurden AP 212 *Electrotechnical design and installation* und AP 221 *Functional data and their schematic representation for process plant* vorgestellt und auf ihre Tauglichkeit hin untersucht, um die Gerätestruktur von Automatisierungssystemen darstellen zu können.

Die Formalisierte Prozessbeschreibung nach VDI/VDE-Richtlinie 3682 besteht aus einem grafischen, formalen Beschreibungsmittel für verfahrenstechnische Prozesse aus

funktionaler Sicht und einem UML-basierten Informationsmodell, das die Syntax und Semantik der Prozessbeschreibung definiert und alle in der Prozessbeschreibung verwendeten Prozessobjekte ausführlich attributiert (Kennzeichnung, Merkmal). Ermöglicht wird so die Integration von ingenieurgemäßer Prozesssicht und informatikorientierter Datenbeschreibung in einem standardisierten universellen und konfigurierbaren (d.h. strukturierbaren und parametrierbaren) Informationsmodell.

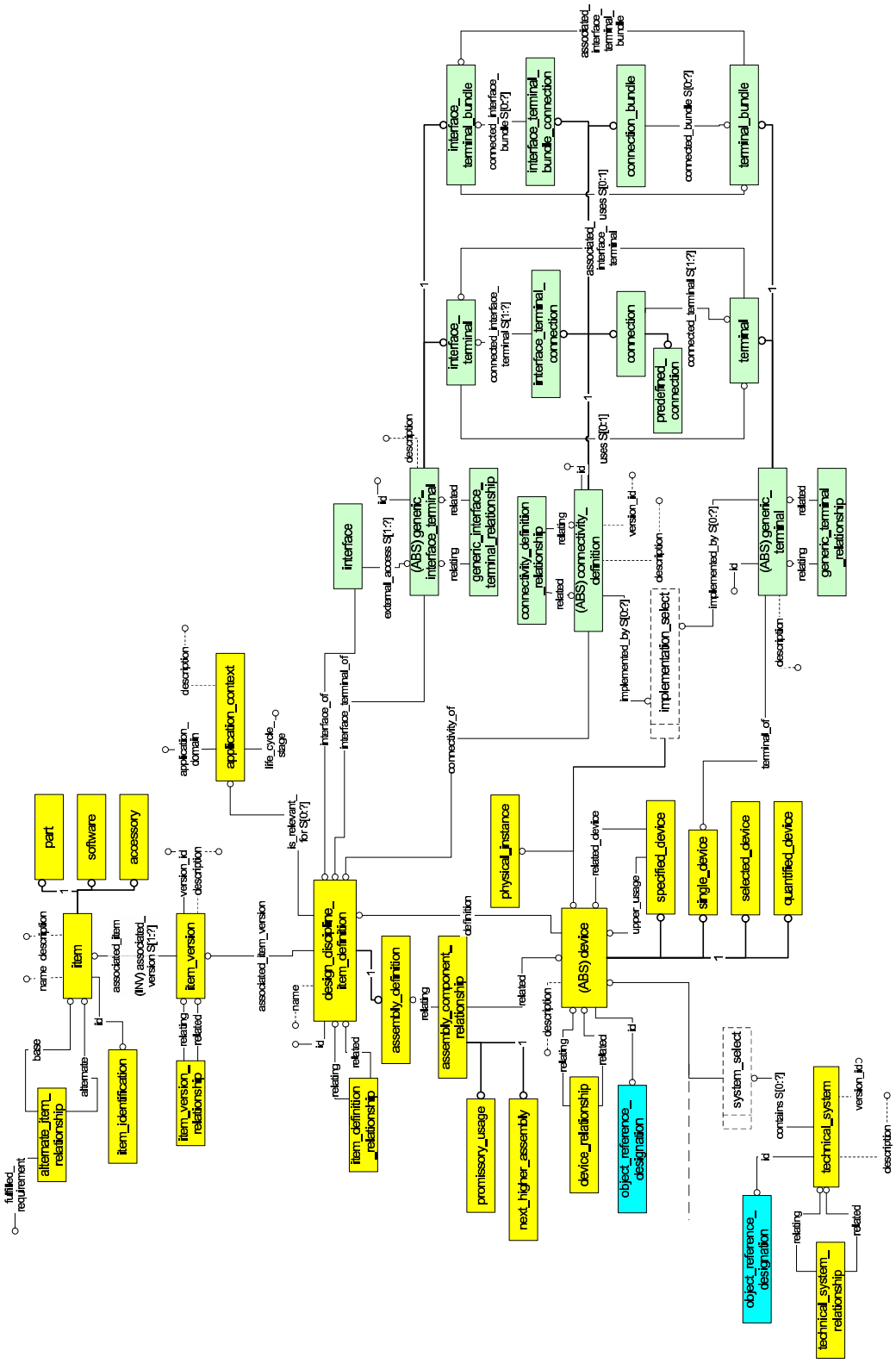


Abbildung 4.12: Das Produktorientierte Teilmodell von AP 212 als EXPRESS-Datenmodell

5 Strukturverträglichkeit automatisierungstechnischer Ontologien

Im Abschnitt 3.1 wurde ein Konzept zur Integration von Ontologien auf der Basis strukturverträglicher Abbildungen vorgestellt. Dieses Integrationskonzept soll nun auf die in Kapitel 4 vorgestellten Beschreibungsmittel und Referenzmodelle angewandt werden. Im nächsten Kapitel wird dann gezeigt, wie dieses Integrationskonzept zur strukturtreuen Modellierung mit Petrinetzen und Bondgraphen herangezogen werden kann.

Wie bereits erwähnt wurde, kombinieren Petrinetz- und Bondgraphenmodelle die rein prozessdynamische Sicht mit einer strukturellen Systemsicht. Es wird nun gezeigt, welche Symmetrien zwischen dieser strukturellen Systemsicht und dem Produktmodell von ISO 10303 herrschen und wie diese formal überprüft werden können. Dazu werden die Modellkonzepte der Informations- und Energieumsetzung (Petrinetze und Bondgraphen) auf der Basis von UML/MOF konzeptualisiert und in Ontologien überführt. Desweiteren werden mit OCL formale Abbildung definiert, die das bereits in Abschnitt 4.3.3 vorgestellte Produktmodell auf die erarbeiteten Petrinetz- und Bondgraphen-Ontologien abbilden. Dabei wird für diese Abbildungen das Kriterium der Strukturverträglichkeit gefordert, so dass es sich hierbei um Morphismen handelt. Die Strukturverträglichkeit führt dazu, dass die im Produktmodell definierten Hierarchie- und Verbindungsrelationen in den Petrinetzen und Bondgraphen erhalten bleiben.

Die Abbildung von Gerätestrukturen auf die Gegenstandsbereiche der Energie- und Informationsumsetzung entspricht dem Entwicklungsprozess der Systemanalyse. Im Gegensatz zur Systemsynthese wird bei der Analyse auf der Grundlage einer Gerätestruktur ein Systemmodell erstellt. Wie man bei der großen Anzahl technischer Baugruppen, Geräte und Bauelemente leicht sieht, sind diese Abbildung nicht injektiv und damit nicht isomorph, so dass die Systemsynthese nur unter kontextabhängiger Einschränkung der Gerätewelt z.B. auf der Basis von Bibliotheken möglich ist.

Das Prinzip der Abbildung von Gerätestrukturen auf Modellkonzepte ist in der Abbildung 5.1 kurz skizziert. Es zeigt ein einfaches elektrisches Netzwerk, das nach dem Produtkmodell von ISO 10303 in Baugruppen aufgeteilt ist, und dessen einzelne Bauteile (Spannungsquelle, Widerstand, Kapazität) seriell bzw. parallel miteinander verschaltet sind. Diese Bauelemente können nun so auf Bondgraphenelemente abgebildet werden, dass die Baugruppen- und Verschaltungsbeziehungen im Bondgraphen erhalten bleiben. Auf der Seite des Bondgraphen werden Baugruppenbeziehungen durch Hierarchisierung

und Verschaltungsbeziehungen durch multiports dargestellt.

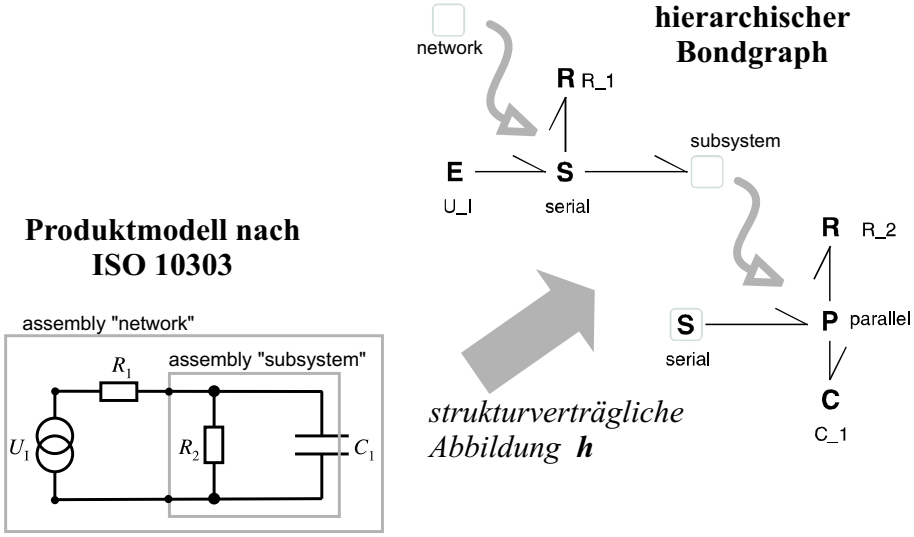


Abbildung 5.1: Strukturvertägliche Abbildung zwischen Produktstruktur und Bondgraph

Nach diesem kurzen einleitenden Beispiel werden nun die Ontologien, deren Strukturvertäglichkeit gezeigt werden soll, näher vorgestellt. Zunächst werden in den ersten Abschnitten die Konzeptualisierungen von Petrinetzen und Bondgraphen erläutert. Beide wurden um Hierarchisierungsmechanismen erweitert, um Baugruppenbeziehungen darstellen zu können. Danach wird das bereits vorgestellte Produktmodell nach ISO 10303, AP 212 aus Abschnitt 4.3.1 auf wesentliche Elemente reduziert und als UML/MOF-Klassendiagramm dargestellt. Dies ist notwendig, da die eigentliche Produktmodelldefinition ja in EXPRESS bzw. EXPRESS-G vorliegt. Den Abschluss des Kapitels bilden die strukturtreuen Abbildungen, die im weiteren auch als Morphismen bezeichnet werden.

5.1 Automatisierungstechnische Ontologien

Grundlage der hier vorgestellten Ontologien sind die Beschreibungsmittel, die in Kapitel 4 vorgestellt wurden. Diese Beschreibungsmittel werden analysiert und dann auf der Basis von MOF konzeptualisiert. Die entstandenen Klassendiagramme können aufgrund der vielen Übereinstimmungen von UML und MOF mit der UML als grafisches Beschreibungsmittel spezifiziert werden. Die Unterschiede zwischen MOF und UML wirken sich ja erst bei der Anwendung von XMI aus (siehe Kapitel 3.2.3).

5.1.1 Petrinetz-Ontologie

Petrinetze sind ein Beschreibungsmittel, um Kommunikationsphänomene in verteilten Systemen darstellen zu können. Ihr großer Vorteil liegt darin, die Verteilung und In-

teraktion informationsverarbeitender Teilsysteme darstellen zu können. Den Petrinetzen liegt also konzeptbedingt neben der reinen Prozesssicht auch eine Strukturansicht zugrunde, auf deren Basis strukturtreue Morphismen definiert werden können. Vor der eigentlichen Diskussion über die Strukturtreue dieser Morphismen wird nun im folgenden eine Ontologie für Petrinetze im Form eines UML/MOF-Klassendiagramms vorgestellt.

Um die für Gerätestrukturen typischen Baugruppenbeziehungen in Petrinetzen wiederfinden zu können, wird ein Hierarchisierungskonzept benötigt. Wir wählen hier in dieser Arbeit das Konzept der verfeinerten oder auch hierarchischen Transitionen, wie es in [Reisig 1985] erläutert wird. Hierbei können transitionsberandete Teilnetze unabhängig von Markierungen und Schaltregeln zu verfeinerten Transitionen zusammengefasst werden. Die zwischen zwei verfeinerten Transitionen liegenden Plätze sind dann gemeinsame Randstellen und repräsentieren Kommunikationskanäle (vgl. S. 81). Diese Kanäle finden sich in verschalteten Gerätestrukturen wieder. Da wir im Produktmodell von ISO 10303 auf die Beschreibung von Verknüpfungsbündeln verzichten (vgl. connection_bundle in Abbildung 4.12 auf S. 107), wird die Verfeinerung von Plätzen nicht benötigt.

Die folgende Abbildung 5.2 zeigt nun eine Konzeptualisierung hierarchischer Petrinetze als Ontologie in Form eines UML/MOF-Klassendiagramms. Neben den Klassen

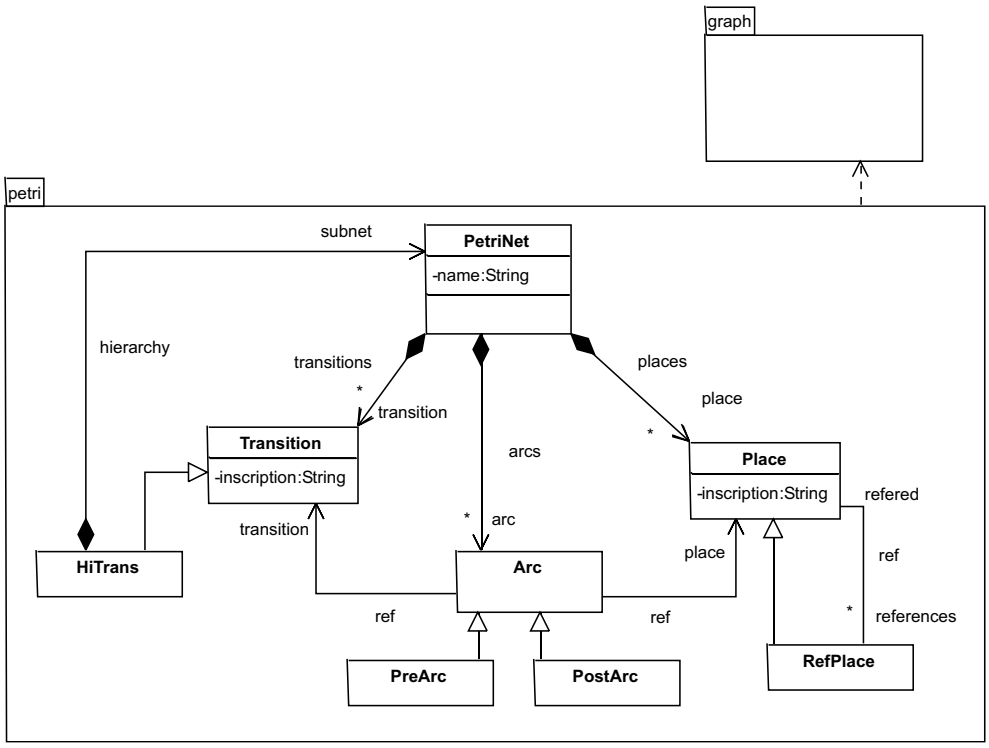


Abbildung 5.2: UML/MOF-Modell der Petrinetz-Ontologie

für die eigentlichen Netzkomponenten (Transition, Place, Arc) enthält Abbildung 5.2 eine Klasse PetriNet, die als Container für die Netzelemente dient. Desweiteren werden Vor- und Nachkanten als Spezialisierung der Klasse Arc unterschieden. Objekte der Klasse

HiTrans sind verfeinerte Transitionen. Das der verfeinerten Transition unterlagerte Unternetz wird als Referenz auf die Klasse PetriNet modelliert. Die Randstellen, an denen die verfeinerte Transitionen an ihre Umwelt angeschlossen sind, erscheinen als Referenzen in den Unternetzen. Diese Referenzplätze sind dann Objekte der Klasse RefPlace. Um als sogenannter *proxy* (Stellvertreter) eines Platzes wirken zu können, sind die Objekte der Klasse RefPlace mit den entsprechenden Place-Objekten über die Referenzen refered und references miteinander verlinkt. Weiterhin zeigt die Abbildung 5.2 eine Abhängigkeit zum graph-Paket, das zur Beschreibung grafischer Eigenschaften von Transition, Plätzen und Kanten dient und hier nicht weiter beachtet wird.

5.1.2 Bondgraphen-Ontologie

Neben den Petrinetzen als Beschreibungsmittel für diskrete, informatische Prozesse werden in der Automatisierungstechnik auch Beschreibungsmittel für kontinuierliche, energetische Prozesse benötigt. Hierzu eignen sich insbesondere Bondgraphen, da diese ähnlich den Petrinetzen über Informationen zur Systemstruktur verfügen. Hierunter fallen z.B. die Verknüpfungsbeziehungen zwischen den einzelnen Modellelementen. Darüber hinaus können durch ein entsprechendes Hierarchiekonzept mit Bondgraphen auch Baugruppenbeziehungen repräsentiert werden. Dieses Hierarchisierungskonzept ist in Anlehnung an [Nebot 1999] konzipiert und fasst Teil-Bondgraphen zu einem neuen Typ von hierarchischen multiports zusammen.

Die Abbildung 5.3 zeigt die Bondgraphen-Ontologie in Form eines UML/MOF-Klassendiagramms. Dieses Klassendiagramm enthält neben den klassischen Bondgraphenelementen auch Elemente zur Hierarchisierung (HiPort, RefNode) und zur Modellierung von Schaltphänomenen (ModulatedTF).

Die beiden zentralen Klassen in Abbildung (5.3) sind *Node* und Bond und repäsentieren die Knoten und Kanten in Bondgraphen. Die Klase *Node* ist abstrakt und in die ebenfalls abstrakten Unterklassen MultiPort, TwoPort und OnePort untergliedert. Dies entspricht der Unterteilung von Bondgraphen in 1-ports, 2-ports und multiports, siehe Tabelle 4.2. Diese Mehrtore wiederum sind untergliedert in die bekannten Bondgraphenelemente wie z.B. Resistor als 1-port, Transformer als 2-port und Parallel als multiport.

Das Hierarchiekonzept wird durch die Klasse HiPort implementiert. Hierarchische Ports (Objekte der Klasse HiPort) sind spezielle multiports, die durch einen Subgraphen verfeinert werden. Alle ports, mit denen ein hierarchischer port in Verbindung steht, werden durch RefNode-Objekte im Subgraph repräsentiert. Desweiteren sind die Bonds, mit denen die hierarchischen Ports in Verbindung stehen, als RefBond-Objekte stellvertretend im Subgraph enthalten. Die RefBond-Objekte sind über Referenzen mit den durch sie vertretenen Bond-Objekten verlinkt und werden benötigt, um bei der Degruppierung von hierarchischen multiports alle Verknüpfungen exakt auflösen zu können.

Wie in Abschnitt 4.2.2 erläutert gibt es eine Reihe unterschiedlicher Konzepte, um schaltende Vorgänge in Bondgraphen zu realisieren. In dieser Arbeit wird das Konzept der modulierten Transformatoren gewählt. Bei modulierten Transformatoren ist

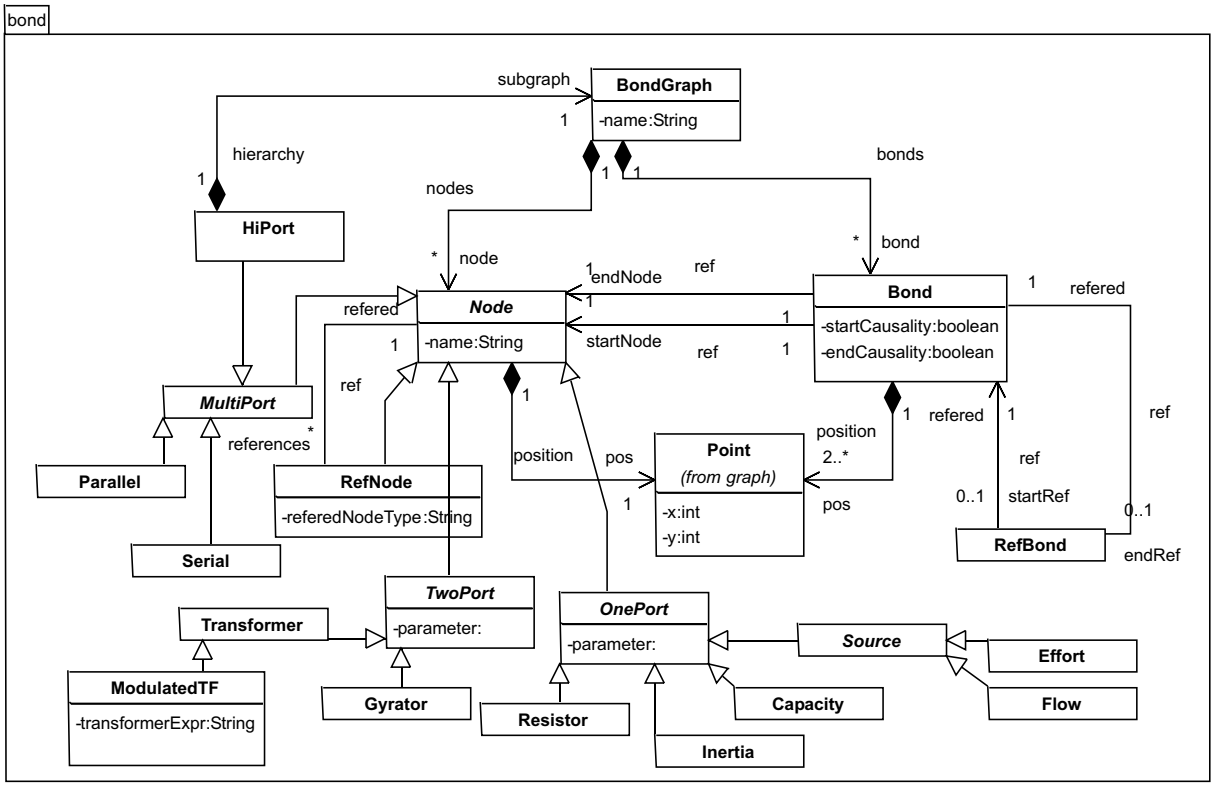


Abbildung 5.3: UML/MOF-Klassendiagramm der Bondgraphen-Ontologie

der Übertragungskoeffizient ein binäre, von aussen gesteuerte Funktion. Dieses Konzept der modulierten Transformatoren wird hier jedoch dahingehend erweitert, dass der Übertragungskoeffizient eine beliebige binäre Funktion über den Eingangs- und Zustandvariablen ist. Diese Konzept wird durch die Klasse ModulatedTF realisiert, deren transformerExpression-Attribut die Funktion des Übertragungskoeffizienten enthält. Die Funktion des Übertragungskoeffizienten kann z.B. ein beliebiger regulärer MATLAB- oder Octave-Ausdruck sein. Formuliert man in dieser Funktion jedoch bewusst arithmetische Ausdrücke über den diskreten Markierungszuständen eines Petrinetzes, kann man auf einfache Weise über modulierte Transformatoren Bondgraphen mit Petrinetzen koppeln, um hybride Prozesse beschreiben zu können. Dies wird im folgenden Kapitel 6 anhand eines Beispiels erläutert.

5.1.3 Produktmodell

Das Produktmodell von ISO 10303 zählt zu den umfassendsten Normenreihen überhaupt und unterstützt mit seinen Anwendungsprotokollen eine Reihe der unterschiedlichsten Anwendungsgebiete wie z.B. die technische Dokumentation, Elektrotechnik, Schiffbau, Verfahrenstechnik und das Gießereiwesen. Wie in Abschnitt 4.3.2.3 gezeigt wurde, eignet sich das Produktmodell der Elektrotechnik, das Anwendungsprotokoll AP 212: *Electro-*

technical design and installation, besonders zur Beschreibung automatisierungstechnischer Geräte/Produkte und wird in dieser Arbeit genutzt. Von besonderem Interesse ist hierbei jedoch nur das produktorientierte Teilmodell, das in Abschnitt 4.3.3 kurz vorgestellt wurde. Dieses Teilmodell wiederum vereint neben Baugruppen- und Verschaltungsinformationen auch Elemente zur Beschreibung von Geräteversionen, -identifikationen und -konfigurationen (z.B. item_version, item_identification, application_context in Abbildung 4.12 auf S. 107), die hier nicht benötigt werden. Desweiteren ist das Produktorientierte Teilmodell in Abbildung 4.12 in EXPRESS-G formuliert und muss für die weitere Nutzung mit XMI sowieso in ein UML/MOF-Klassendiagramm umgewandelt werden.

Für die Umwandlung des Datenmodells von AP 212 in UML kann man [ISO 2001] anwenden. Diese Abbildung der Sprachkonstrukte von EXPRESS auf UML läßt sich aber nur indirekt aus den XMI-Fragementen herleiten. Desweiteren ergeben sich Probleme bei Aggregationstypen (Listen, Mengen, Multimengen, Arrays) und *select*-Auswahltypen, für die implementierungsunfreundliche, umständliche *mappings* angeboten werden. Weiterer Kritikpunkt ist, dass in dieser Norm Pakete nicht unterstützt werden. Für die Darstellung des Produktmodells von ISO 10303 wird deshalb ein eigenes anwendungs- und implementierungsfreundliches *mapping* verwendet.

Die folgende Abbildung zeigt die baugruppen- und verschaltungsrelevanten Elemente des produktorientierten Teilmodells von ISO 10303, AP 212 in Form eines UML/MOF-Klassendiagramms. Dieses Klassendiagramm dient im weiteren Verlauf dieser Arbeit als die produktdefinierende Ontologie.

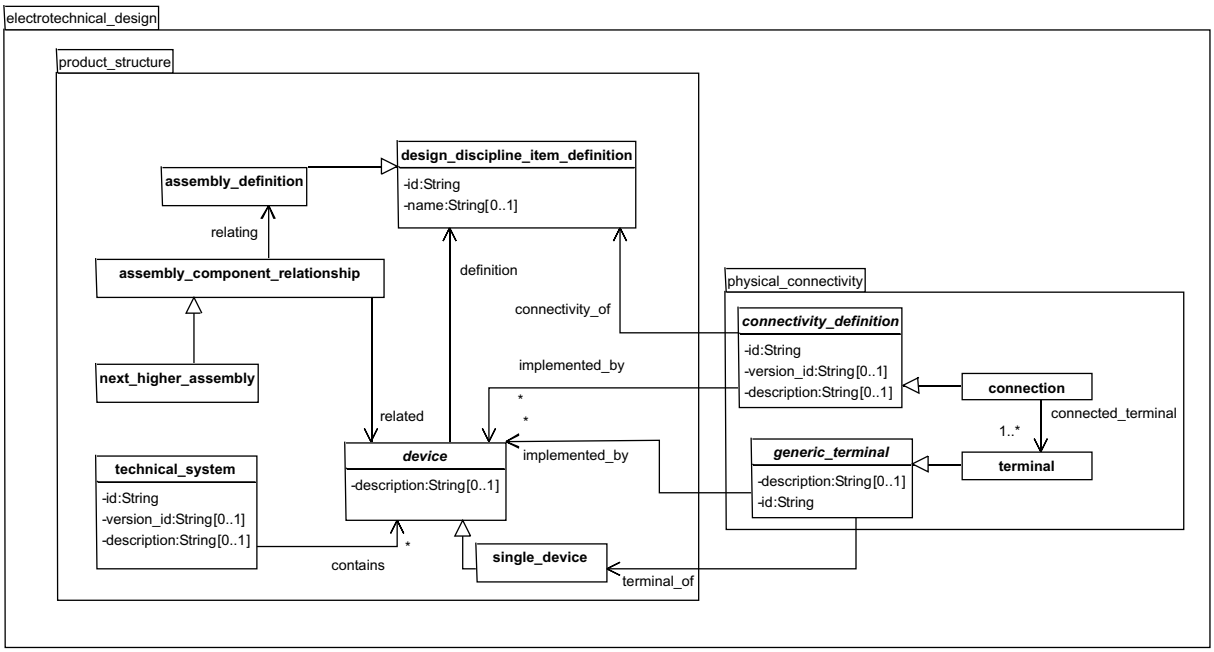


Abbildung 5.4: UML/MOF-Modell der Produkt-Ontologie

Die beiden Pakete product_structure und physical_connectivity entsprechen den gleich-

namigen UoF's aus dem Anwendungsprotokoll AP212. Beide Pakte sind eingebettet in das übergeordnete Paket electrotechnical_design, das damit nach XMI den *root-node* produktdefinierender XML-Dateien bildet.

Zentrales Element des Paketes product_structure ist die abstrakte Klasse device. Einzelne Geräte oder Baugruppen sind Instanzen der von device abgeleiteten Klasse single_device. Geräte, bei denen es sich um hierarchisch aufgebaute Baugruppen handelt, zeigen mit ihrer definition-Referenz auf assembly_definition-Objekte. Der Aufbau einer Baugruppe wird durch Objekte der Klasse assembly_component_relationship definiert, die mit ihrer related-Referenz wiederum auf die in der Baugruppe enthaltenen single_device-Objekte zeigen. Mit diesem Mechanismus können komplexe Baugruppenhierarchien erschaffen werden.

Das Paket physical_connectivity enthält mit den Klassen connection und terminal die geeigneten Elemente zur Darstellung von Verknüpfungen/Verschaltungen zwischen einzelnen Geräten. Dies geschieht, indem die connection-Klasse instanziert wird und diese Instanzen diejenigen terminal-Objekte referenzieren, deren terminal_of-Referenzen auf die Geräte zeigen, die an einer Verknüpfung teilnehmen.

Das reduzierte Produktmodell aus Abbildung 5.4 enthält damit den minimalen Satz an Klassen, um Baugruppenhierarchien und Geräteverknüpfungen darstellen zu können.

5.2 Strukturverträglichkeit

Nachdem nun Petrinetze, Bondgraphen und das AP212-Produktmodell mit UML/MOF einheitlich konzeptualisiert und in Ontologien überführt wurden, kann nun deren Strukturverträglichkeit untersucht werden. Dies erfordert es, strukturtreue Abbildungen zwischen diesen Ontologien, sogenannte Morphismen, zu finden. Im folgenden werden nun diese Morphismen als Abbildungen vom Produktmodell in die Petrinetze und Bondgraphen in Form von OCL-*constraints* spezifiziert. Des weiteren werden Relationen in diesen Ontologien definiert, die bei Anwendung der Morphismen strukturtreu erhalten bleiben. Diese Relationen beziehen sich auf die im reduzierten Produktmodell in Abbildung 5.4 dargestellten Baugruppen- und Verschaltungsbeziehungen, die zwischen Geräten und/oder Baugruppen bestehen. Diese Morphismen können, wie im nächsten Abschnitt gezeigt, im Prozess der Systemanalyse zur konsistenten, strukturtreuen Modellbildung genutzt werden.

5.2.1 Petrinetz-Morphismus

Im folgenden wird anhand des Klassendiagramms in Abbildung 5.5 ein Morphismus spezifziert, der das reduzierte Produktmodell von ISO 10303, AP 212 aus Abbildung 5.4 auf die Petrinetz-Ontologie in Abbildung 5.2 strukturverträglich abbildet.

Grundlage dieses Morphismus ist die Abbildung von Geräten auf Transitionen. Wie ja bereits im Kapitel 4.1 ausführlich erläutert wurde, versinnbildlichen die Transition die

aktiven, gerätetechnisch umsetzbaren Schaltelemente. Die Plätze hingegen etablieren die Kommunikationskanäle und dienen deshalb zur Definition von Verknüpfungsbeziehungen.

Die Spezifikation der einzelnen Morphismen beruht auf Klassendiagrammen. Die Baugruppen- und Verknüpfungsrelationen, bezüglich derer die Morphismen strukturtreu sind, werden mit OCL-*constraints* definiert.

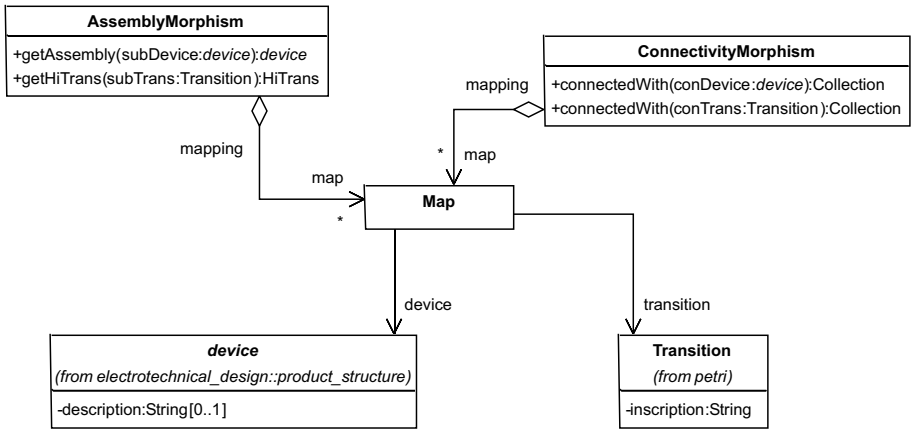


Abbildung 5.5: Klassendiagramm des Petrinetz-Morphismus

Die Abbildung 5.5 zeigt das Klassendiagramm des Petrinetz-Morphismus'. Im Zentrum des dort gezeigten Klassendiagramms steht die Klasse Map. Objekte dieser Klasse zeigen auf *device*-Objekte aus dem Paket electrotechnical_design und auf Transition-Objekte aus dem Paket petri. Mit diesen Objektpaaren bildet die Klasse Map die eigentliche Abbildung von der Gerätewelt in die Petrinetzwelt. Die Paarbildung basiert auf einer simplen Namensgleicheit, die durch folgenden OCL-*constraint* definiert ist:

```
context Map
   inv descriptionExistence :
      device.description->notEmpty

   inv sameName :
      device.description = transition.inscription
```

In der ersten Invariante wird gefordert, dass jedes device-Objekt ein nicht-leeres description-Attribut besitzt, da dieses Attribut gemäß AP 212 ja optional ist und deshalb durchaus leer sein kann. Die zweite Invariante fordert die Gleichheit der Attribute description der Klasse device und inscription der Klasse Transition.

Die Morphismen werden durch die Klassen AssemblyMorphism und ConnectivityMorphism repräsentiert. Die dem Morphismus zugrundeliegende Abbildung von der Produkt- in die Petrinetzwelt wird durch eine Menge von Map-Objekten realisiert. Jede Morphismus-Klasse besitzt darüber hinaus zwei Relationen, bezüglich derer die Morphismen strukturverträglich sind. Beide Morphismen-Klassen werden nun näher erläutert.

Baugruppenmorphismus

Die Klasse AssemblyMorphism repräsentiert den für die Baugruppenrelationen strukturtreuen Morphismus. Diese Strukturtreue läßt sich in OCL folgendermaßen formulieren:

```
context AssemblyMorphism
   inv Homomorphism:
      map->forAll(m1, m2 |
         m2 <> m1 and m2.device = getAssembly(m1.device)
            implies
         m2.transition = getHiPort(m1.transition)
      )
```

Dieser *constraint* ist die Umsetzung der Gleichung (3.30) auf Seite 57. Jedoch stellt dieser *constraint* den Spezialfall einer binären Relation dar und wird über einen Funktionenformalismus spezifziert. Gleichung (3.30) hingegen wird ja über Relationen spezifziert.

In der Invariante wird gefordert, dass zwischen Transitionen Hierarchiebeziehungen gelten müssen, falls zwischen den entsprechenden Elementen in der Gerätewelte vergleichbare Baugruppenbeziehungen herrschen. Die Baugruppenbeziehung der Gerätewelt ist wie folgt definiert:

```
context AssemblyMorphism::getAssembly(subDevice : Device) : Device
   let devices : Collection = device.allInstances
   let assemblies : Collection = assembly_component_relationship.allInstances in

   pre subInAssembly :
      assemblies->exists(as | as.related = subDevice)
   post getAssembly :
      result =
         devices->select(dev |
            assemblies->exists(  as |
               (as.relating = dev.definition) and (as.related = subDevice)
            )
         )
   post uniqueness
      result.size == 1
```

In der Vorbedingung `pre subInAssembly`, wird gefordert, dass das Gerät `subDevice` überhaupt in einer Baugruppe verbaut ist. In der ersten Nachbedingung (`post getAssembly`) wird die Menge der Geräte ermittelt, die `subDevice` enthalten. Da jedes Geräte nur in genau einer Baugruppe verbaut sein kein, wird in der zweiten Nachbedingung (`post uniqueness`) überprüft, ob diese Menge die Mächtigkeit 1 besitz. Somit handelt es sich bei der Baugruppenrelation um eine binäre Relation und damit um eine echte Abbildung.

Auf ähnliche Weise ist die Baugruppenbeziehung in der Petrinetz-Ontologie definiert. Hier besteht diese Beziehung zwischen einer verfeinerten Transition und all den Transitionen, die das Subnetz der verfeinerten Transition enthält.

```
context AssemblyMorphism::getHiTrans(subTrans : Transition) : HiTrans
   let allHiTrans : Collection = HiTrans.allInstances in

   pre subTransInHierarchy :
      allHiTrans->exists( hi | hi.subnet.transitions->contains (subTrans) )
```

```
   post getHiTrans :
      result = allHiTrans->select( hi | hi.subnet.transitions->contains (subTrans) )

   post uniqueness
      result.size == 1
```

Während in der Vorbedingung wieder gefordert wird, dass die Transition `subTrans` überhaupt Teil einer Hierarchiebeziehung ist, gibt die erste Nachbedingung die Menge der verfeinerten Transitionen zurück, die `subTrans` enthalten. Auch diese Menge muss die Mächtigkeit 1 besitzen, da jede Transition nur in jeweils einer hierarchischen Transition enthalten sein kann.

Verknüpfungsmorphismus

Mit der Klasse ConnectivityMorphism in Abbildung 5.5 wird der bezüglich der Verknüpfungsrelationen strukturtreue Morphismus modelliert. Dieser bildet Geräte auf Transition derart ab, dass Verknüpfungen in der Gerätewelt im Petrinetz erhalten bleiben. Während sich die Verknüpfungen der Gerätewelt in Instanzen der Klasse `connection` manifestieren, geschieht dies bei den Petrinetzen durch gemeinsame Randstellen, siehe Seite 81.

Mit der Verknüpfungsoperation `connectedWith`, die sowohl für das Produktmodell als auch für Petrinetze defniniert werden kann, ist die Strukturtreue nun folgendermaßen formalisiert werden:

```
context ConnectivityMorphism
   inv Homomorphism:
      map->forAll(m1, m2 |
         m2 <> m1 and connectedWith(m1.device)->constains(m2.device)
            implies
         connectedWith(m1.transition)->contains(m2.transition)
      )
```

Da ein Bauteil mit mehreren anderen Bauteilen verknüpft werden kann, sind Verknüpfungsrelationen im Gegensatz zu Baugruppenrelationen mehrwertig. Dementsprechend hat die Ergebnismenge der `connectedWith`-Operation eine Mächtigkeit > 1.

Die Verknüpfungsrelation ist in der Gerätewelt wie folgt definiert:

```
context ConnectivityMorphism::connectedWith(conDevice : device) : Collection
   let devices : Collection = device.allInstances
   let connections : Collection = connection.allInstances in

   pre conDeviceConnected :
      connections->exists( con |
         con.connected_terminal->contains( term |
            term->terminal_of = conDevice )
         )
   post : result =
      devices->select( dev |
         connection->exists( con |
```

```
            con.connected_terminal->contains( term |
               term.terminal_of = dev)
                        and
            con.connected_terminal->contains( term |
               term.terminal_of = conDevice)
         )
      )
```

Um die Menge der mit einem gegebenen Geräte `conDevice` verknüpften anderen Geräte zu ermitteln, müssen alle Instanzen der Klasse `connection` innerhalb einer Produktstruktur überprüft werden. Die in diesen `connection`-Instanzen durch das `terminal_of`-Attribut referenzierten `terminal`-Instanzen sind Geräteschnittstellen und spezifizieren die Menge der miteinander verbundenen Geräte.

Auf der Seite der Petrinetze werden Verknüpfungsrelationen zwischen den Transitionen durch gemeinsame Vor- bzw. Nachplätze hergestellt.

```
context ConnectivityMorphism::connectedWith(conTrans: Transition) : Collection
   let transitions : Collection = Transition.allInstances
   let places : Collection = Places.allInstances
   let arcs : Collection = arc.allInstances

   pre conTransConnected :
      places->exists ( place |
         arcs->exists( arc |
            ( arc.transition = conTrans and arc.place = place )
         )
      )
   post :
      result =
      trans->select( resultTrans |
         places->exists( place |
            arcs->exists( a1 |
               a1.transition = conTrans and a1.place = place
            )
            and
            arcs->exists( a2 |
               a2.transition = resultTrans and a2.place = place
            )
         )
      )
```

Zunächst muss eine Transition über eine Kante mit einem Platz verbunden sein, um überhaupt an einer Verknüpfung teilhaben zu können, siehe `pre conTransConnected`. Zwei Transitionen sind dann miteinander verknüpft, wenn es einen gemeinsamen Platz in deren Vor- oder Nachbereich gibt. Die Menge der Transitionen, die mit einer gegebenen Transition `conTrans` verknüpft sind, wird in der Nachbedingung des *constraints* ermittelt und hat im Allgemeinen eine Mächtigkeit > 1.

Beispiel

Die oben erwähnten strukturtreuen Abbildungen zwischen dem Produktmodell nach ISO 10303 und der Petrinetz-Ontologie in Abbildung 5.2 werden nun anhand einer einfachen Ampelanlage erläutert. Insbesondere wird aufgezeigt, welche Auswirkungen die Einhaltung der Strukturtreu auf die Modellbildung mit Petrinetzen hat.

Die folgende Abbildung 5.6 zeigt die Produktstruktur einer einfachen Ampelanlage bestehend aus zwei Ampelbaugruppen (*Ampel 1* und *Ampel 2*) mit jeweils drei Lampen und einem Steuergerät (*Steuerung*) mit Steueralgorithmus. Der Steueralgorithmus wiederum besteht aus zwei Programmeinheiten (z.B. SPS-Netzwerk), die die Steuerung übernehmen. Die Programmeinheiten werden hier ebenfalls als Bauteil im weitesten Sinne verstanden. Dies stellt eine nicht sehr naheliegende Abstraktion dar, ist aber durchaus im Sinne von ISO 10303, AP 212, da dort alle Bauteile auf oberster Taxonomieebene in Geräte und Software unterteilt werden. Der Steueralgorithmus könnte aber auch über zwei dezentrale, physikalisch getrennte Steuergeräte verteilt sein.

Neben der Bauteilhierarchie zeigt die Abbildung 5.6 auch eine Verknüpfungsstruktur. Die Steuerung ist extern mit den Ampelständern und intern zwischen den Programmeinheiten verknüpft. Sinn der nachfolgenden Modellbildung mit Petrinetzen ist

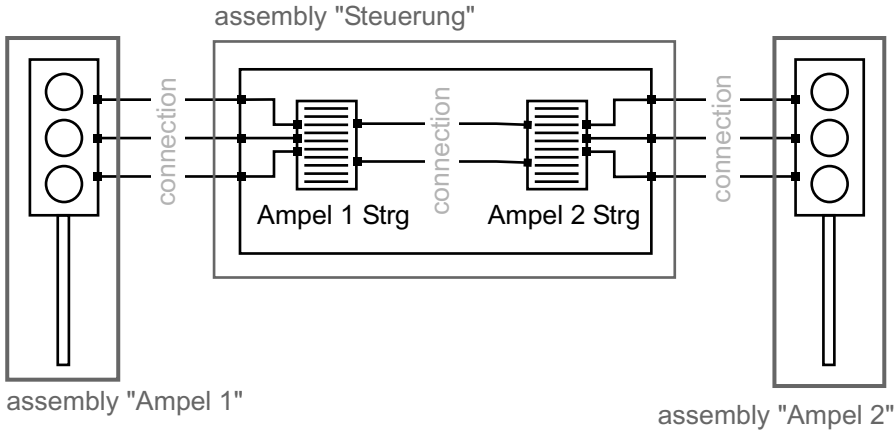


Abbildung 5.6: Produktstruktur einer Ampelanlage

es, die oben erwähnte Baugruppen- und Verknüpfungsstruktur strukturtreu beizubehalten. Dazu ist das Petrinetz in Abbildung 5.7 auf oberster Ebene in drei verfeinerte Transition partioniert, die die Baugruppen *Ampel1*, *Ampel 2* und *Steuerung* repräsentieren. Die Verschaltung der Anlage wird durch Plätze modelliert, die die einzelnen Baugruppen-Transitionen miteinander verknüpfen.

Die Ampelbaugruppen wiederum sind in drei Transitionen unterteilt, um die Ampelleuchten strukturtreu zu modellieren. Die Plätze mit eingefasstem Dreieck stellen Referenzplätze dar.

Bei den üblichen Verfahren der Modellbildung mit Petrinetzen wird das Streckenmodell u. U. vernachlässigt. Die Lampen würde man dann als Stellen modellieren und mit

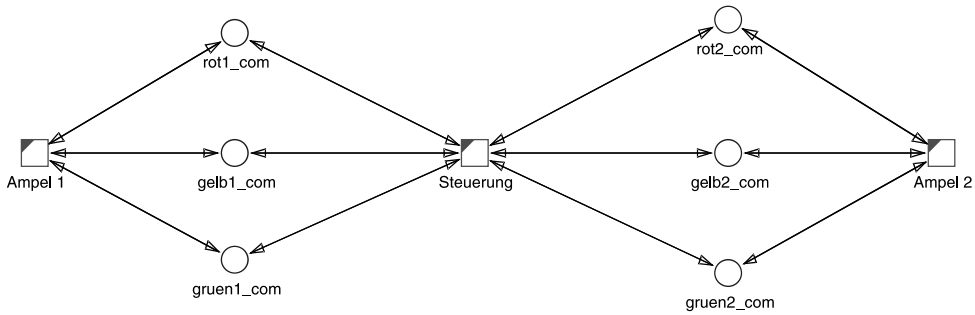


Abbildung 5.7: Strukturtreues Petrinetz der Ampelanlage

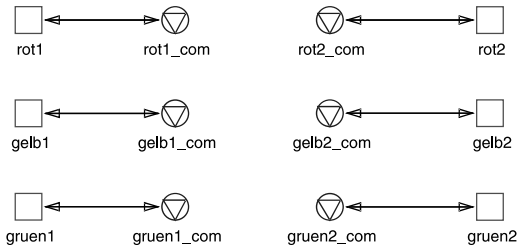


Abbildung 5.8: Petrinetz der Ampeln *1* und *2*

den Plätzen rot1_com usw. identifizieren. Dies ist aus Sicht der Steuerungsspezifikation durchaus möglich, jedoch eine grobe Abstraktion, da Lampen ja automatisierungstechnische Aktoren mit nichttrivialem Verhalten sind und die Modellierung mit verfeinerten Transitionen erfordern. Dies zeigt sich spätestens bei der Modellierung zuverlässigkeitsrelevanter oder dynamischer Phänome wie Ausfall, Wartung, Reparatur, Latenzzeiten usw.

Wie bereits erwähnt ist die Steuerung in zwei Programm- oder auch Steuereinheiten unterteilt (*Ampel 1 Strg* und *Ampel 2 Strg*). Dies spiegelt sich im Petrinetz in Abbildung 5.9 wider. Um beide Ampelzyklen koordinieren zu können, sind die Programmeinheiten

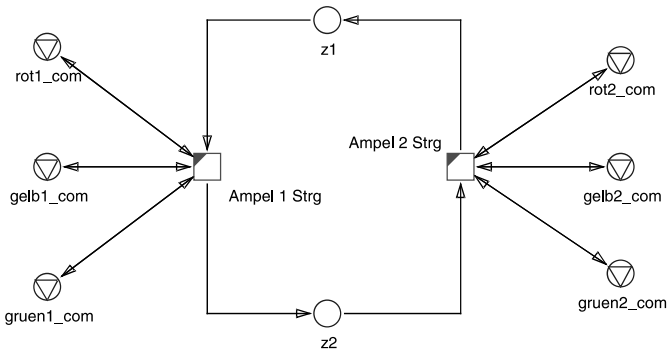


Abbildung 5.9: Petrinetz der Ampelsteuerung

über Plätze miteinander verknüpft. Eine entsprechende programmtechnische Implemen-

tierung wären z.B. Merker in SPS-Programmen.

Der eigentliche Steueralgorithmus für die Anlage kann in Produktstrukturbildern wie Abbildung 5.6 nicht abgebildet werden. Hier muss nun im Petrinetz der Sollprozess modelliert werden. Eine mögliche Lösung für die beiden Steueralgorithmen der Ampelsteuerung (vgl. [v. Aspern 1994b]) zeigt die folgende Abbildung 5.10.

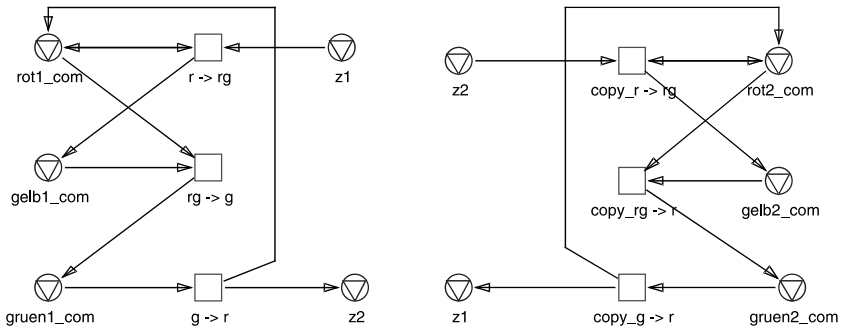


Abbildung 5.10: Petrinetz der Steueralgorithmen

Durch die strukturtreue Modellbildung können im Petrinetz sowohl die steuerungstechnischen als auch die produktstrukturorientierten Aspekte der Ampelanlage beschrieben werden.

5.2.2 Bond-Morphismus

Ähnlich wie bei Petrinetzen in Abschnitt 5.2.1 kann man für Bondgraphen einen Morphismus formulieren, der das Produktmodell von ISO 10303 strukturtreu in die Bondgraphen abbildet. Auch hier bezieht sich die Strukturtreue auf Baugruppen- und Verknüpfungsrelationen. Im Gegensatz zum Petrinetz-Morphismus, bei dem die Geräte ja nur auf die Transitionen abgebildet wurde, werden beim Bondgraphen-Morphismus Geräte des Produktmodells auf unterschiedliche 1-port bzw. 2-port Knoten wie z.B. Widerstände, Transformatoren, *effort*-Quellen usw. abgebildet. Die s- und p-Verknüpfungen hingegen entsprechen keinen Geräten und werden nur für die Definition von Verknüpfungsrelationen herangezogen. Baugruppenbeziehungen in der Gerätewelt werden im Bondgraphen durch Hierarchisierung dargestellt.

Das folgende Klassendiagramm in Abbildung 5.11 zeigt nun die beiden Baugruppen- und Verknüpfungsmorphismen. Die Klasse Map vermittelt die Abbildung von Geräten der Klasse device auf Bondgraphenknoten, repräsentiert durch die abstrakte Klasse Node. Die Paarbildung beruht auch bei diesem Morphismus auf Namensgleichheit.

```
context Map
   inv descriptionExistence :
      device.description->notEmpty
   inv sameName :
      device.description = node.name
```

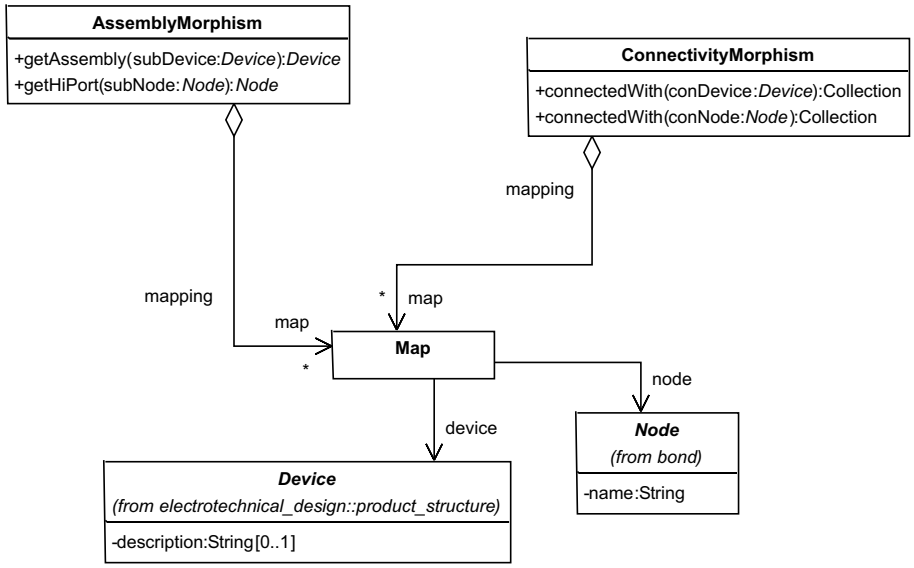


Abbildung 5.11: Klassendiagramm des Bondgraphen-Morphismus

Der *constraint* bedeutet, dass Geräte und die jeweiligen Knoten im Bondgraphen den gleichen Namen tragen.

Der Baugruppenmorphismus und der Verknüpfungsmorphismus werden nun näher vorgestellt.

Baugruppenmorphismus

Der Baugruppenmorphismus wird durch die Klasse AssemblyMorphism spezifziert. Die Relationen, bezüglich derer der Morphismus strukturtreu ist, sind die Methoden getAssembly und getHiPort. Die Strukturtreue folgt aus dem folgenden *constraint*.

```
context AssemblyMorphism
   inv Homomorphism:
      map->forAll(m1, m2 |
         m2 <> m1 and m2.device = getAssembly(m1.device)
            implies
         m2.node = getHiPort(m1.node)
```

Die Baugruppenrelation `getAssembly` ist identsich zu der auf Seite 116 und wird nicht weiter behandelt. Auf Seiten der Bondgraphen werden Baugruppenrelationen durch Hierarchiebeziehungen etabliert. Die Hierarchiebeziehung findet sich in der Methode getHiPort, die wie folgt spezifiziert ist.

```
context AssemblyMorphism::getHiPort(subNode : Node) : Node
   let hiPorts : Collection = HiPort.allInstances in

   pre subNodeInHierarchy :
      hiPorts->exists( hi | hi.subgraph.nodes->contains (subNode) )
```

```
   post getHiPort :
      result = hiPorts->select( hi | hi.subgraph.nodes->contains (subNode) )

   post uniqueness
      result.size == 1
```

Da jeder Bondgraphenknoten sich nur in einem hierarchischen Knoten befinden kann, ist die Mächtigkeit der Ergebnismenge 1.

Verknüpfungsmorphismus

Der Verknüpfungsmorphismus zwischen der Geräte- und Bondgraphenwelt findet sich in der Klasse `ConnectivityMorphism` wieder, siehe Abbildung 5.11. Die Klasse bezieht sich - analog zum Verknüpfungsmorphismus bei Petrinetzen - auf die `connectedWith`-Relationen und erfüllt das folgende Strukturtreuekriterium:

```
context ConnectivityMorphism
   inv Homomorphism:
      map->forAll(m1, m2 |
         m2 <> m1 and connectedWith(m1.device)->constains(m2.device)
            implies
         connectedWith(m1.node)->contains(m2.node)
```

Die Verknüpfungsrelation auf Seiten des Produktmodells ist identisch zu der in Kapitel 5.2.1 auf Seite 117.

Die Verknüpfungsrelation für Bondgraphen stützt sich auf deren Eigenschaft, Verschaltungen zwischen mehreren Knoten durch multiports zu modellieren. Damit gelten zwei Knoten als verknüpft, falls sie an demselben s- bzw. p-Knoten angeschlossen sind. Dies kann folgendermaßen spezifiziert werden:

```
context ConnectivityMorphism::connectedWith(conNode : Node) : Collection
   let nodes : Collection = Node.allInstances
   let bonds : Collection = Bond.allInstances in

   pre conNodeNoMultiPort :
      not (conNode.oclIsTypeOf(Serial) or conNode.oclIsTypeOf(Parallel))

   pre conNodeConnected :
      nodes->exists( multiport |
            ( multiport.oclIsTypeOf(Parallel) or multiport.oclIsTypeOf(Serial) )
         and
            ( bonds->exists( b |
               ( b.endNode = conNode and b.startNode = multiport) or
               ( b.startNode = conNode and b.endNode = multiport)
            )
      )
   post connectedWith :
      result =
      nodes->select( resultNode |
         nodes->exists( multiport |
               ( multiport.oclIsTypeOf(Parallel) or multiport.oclIsTypeOf(Serial) )
```

```
            and
               bonds->exists( b1 |
                  ( b1.endNode = conNode and b1.startNode = multiport) or
                  ( b1.startNode = conNode and b1.endNode = multiport)
               )
               and
               bonds->exists( b2 |
                  ( b2.endNode = resultNode and b2.startNode = multiport) or
                  ( b2.startNode = resultNode and b2.endNode = multiport)
               )
         )
      )
   post resultNoMultiPort :
      result->select(resultNode |
         not (resultNode.oclIsTypeOf(Serial) or resultNode.oclIsTypeOf(Parallel))
      )
```

Die Vorbedingung `conNodeConnected` stellt sicher, dass ein gegebener Knoten `conNode` auch mit einem multiport verknüpft ist. Die Nachbedingung `connectedWith` ermittelt für einen gegebenen Knoten `conNode` die Menge all der Knoten, die mit `conNode` über einen gemeinsamen multiport verknüpft sind. Dazu müssen Kanten existieren, die alle Knoten der Ergenbismenge plus den Knoten `conNode` mit diesem multiport verbinden.

Falls zwei 1-port-Knoten über einen 2-port-Knoten (Transformator, Gyrator) miteindander verbunden wären, würde nach obiger Definition keine Verknüpfungsrelation bestehen, da Verknnüpfungen nur über multiport-Knoten bestehen. Dies ist sinnvoll, da Transformatoren ja gerätetechnische Einrichtungen sind und nicht wie s- und p-Knoten Leitungsstränge oder Netzwerke darstellen.

Wie bereits erwähnt sollten sinnvollerweise Geräte aus dem Produktmodell nur auf 1-ports, 2-ports und hierarchische ports abgebildet werden. Deshalb muss wenn möglich auch innerhalb der Relation `connectedWith(conNode : Node)` ausgeschlossen werden, dass s- oder p-Knoten mit anderen Knoten als verknüpft gelten. Dies wird durch geeignete Typüberprüfungen in den Vor- und Nachbedingungen `conNodeNoMultiPort` und `resultNoMultiPort` sichergestellt, sodass s- und p-Knoten aus dem Definitions- und Wertebereich von `connectedWith(conNode : Node)` ausgeschlossen sind.

Beispiel

Anhand eines kleinen Beispielnetzwerkes soll die Strukturverträglichkeit von Produktdaten und Bondgraphen näher erläutert werden. Die strukturtreue Modellbildung wird durch Bondgraphen ja schon vom Konzept her unterstützt und kann deshalb ganz einfach durchgeführt werden.

Die folgende Abbildung 5.12 zeigt die Produktstruktur eines passiven elektrischen Netzwerks, das aus einer Spannungsquelle, zwei Widerständen und einem Kondensator besteht. Die Bauteile sind in Baugruppen organisiert (network und subsystem) und über connection-Instanzen seriell bzw. parallel verschaltet. Da es für die elektrischen Bauelemente in der Bondgraphentheorie bereits entsprechende Knotentypen gibt, muss bei der

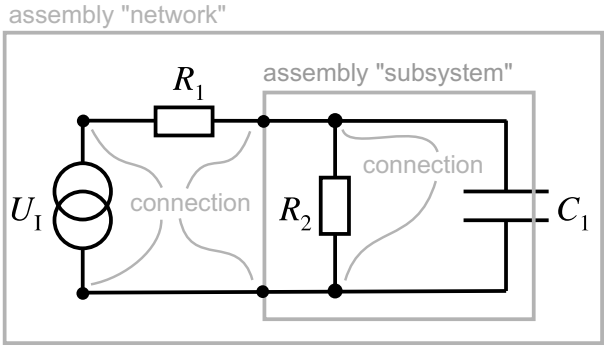


Abbildung 5.12: Produktstruktur eines einfachen elektrischen Netzwerks

Modellbildung nur noch auf Namensgleichheit geachtet werden. In der Abbildung 5.13

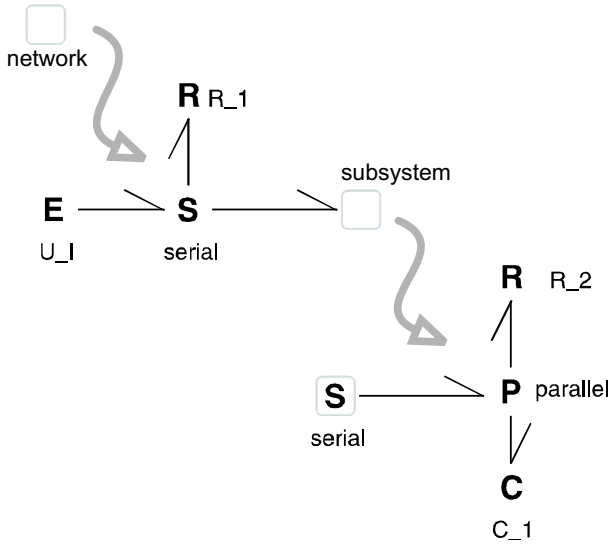


Abbildung 5.13: Hierarchischer Bondgraph eines einfachen elektrischen Netzwerks

ist links der Bondgraph der obersten Hierarchiestufe dargestellt. Er beinhaltet einzig und allein den hierarchischen Knoten network, der der namensgleichen Baugruppe in Abbildung 5.12 entspricht. Die Verfeinerung dieses hierarchischen Knotens ist in der Mitte der Abbildung 5.13 dargestellt. Dieser Teilgraph enthält die Spannungsquelle U_I, den Widerstand R_1 und die Unterbaugruppe subsystem. Die Verknüpfungen werden hier durch die s-Verknüpfung serial modelliert. Im unteren rechten Teil von Abbildung 5.13 ist schließlich der Bondgraph der untersten Hierarchiestufe mit dem Widerstand R_2 und dem Kondensator C_2 dargestellt. Er repräsentiert die Baugruppe subsystem. Die in dieser Baugruppe herrschende Verknüpfung ist mit dem p-Knoten parallel modelliert.

Nach erfolgter Modellbildung kann aus dem gezeigten Bondgraphen nun eine Differenzialgleichungssystem erzeugt werden, mit dem die Prozessdynamik des Netzwerks aus Abbildung 5.12 untersucht werden kann. Somit können mit Bondgraphen ähnlich zu Pe-

trinetzen sowohl die strukturelle Sicht als auch die dynamische Sicht eines technischen Systems modelliert werden.

5.3 Zusammenfassung

Nachdem im Abschnitt 4 verschiedene Beschreibungsmittel und Referenzmodelle der Automatisierungstechnik vorgestellt wurden, war es das Ziel dieses Kapitels, das Ontologieintegrationskonzept der Strukturverträglichkeit aus Abschnitt 3.1.3.2 auf diese Modellkonzepte anzuwenden.

Zuerst wurden die Beschreibungsmittel, soweit dies bei Petrinetzen und Bondgraphen noch notwendig war, in Form von UML/MOF-Modellen als Ontologien konzeptualisiert (Abschnitt 5.1.1 und 5.1.2) bzw. das bereits bestehende Produktmodell überarbeitet, gekürzt und ebenfalls in UML/MOF-Form gebracht (Abschnitt 5.1.3).

Desweiteren wurden Morphismen definiert, die das Produktmodell auf die Petrinetzontologie (Abschnitt 5.2.1) und auf die Bondgraphenontologie (Abschnitt 5.2.2) abbilden. Diese Morphismen sind strukturtreu bezüglich Baugruppen- und Verknüpfungsrelationen und wurden mit OCL formal spezifiziert.

Anhand einfacher Beispiele wurden die Morphismen nochmals kurz erläutert. Insbesondere wurde anhand dieser Beispiele deutlich, welche Auswirkungen die Strukturverträglichkeit für die Modellbildung und Systemanalyse mit Petrinetzen und Bondgraphen hat.

6 Implementationskonzept und Anwendung

Nachdem im vorigen Kapitel die Strukturverträglichkeit von Petrinetzen und Bondgraphen mit dem Produktmodell nach ISO 10303 spezifiziert wurde, soll nun gezeigt werden, wie diese Spezifikation softwaretechnisch umgesetzt werden kann, um so eine Werkzeugumgebung zu bieten, die das strukturtreue Modellieren mit Petrinetzen und Bondgraphen ermöglicht. Inwieweit die Strukturverträglichkeit Auswirkungen auf die Modellbildung hat, wurde bereits in den Modellierungsbeispielen auf Seite 119 und 124 gezeigt.

Die vorgestellte prototypische Werkzeugumgebung unterstützt neben dem strukturtreuen Modellieren mit Petrinetzen auch die Simulation kombinierter Petrinetz-/Bondgraphenmodelle, um die im Abschnitt 2.6 beschriebenen hybriden Systeme untersuchen zu können.

6.1 Werkzeugumgebung

Die Abbildung 6.1 auf Seite 128 zeigt nun die Architektur dieser eingangs erwähnten Werkzeugumgebung.

Im Zentrum steht das unter Java programmierte Softwarepaket `symmetry`. Es liest Petrinetz- und Bondgraphenmodelle in Form von XML-Dateien ein und vergleicht diese mit einer gegebenen Produktstruktur, die ebenfalls im XML-Format vorliegt. Die den XML-Dateien zugrundeliegenden XML-Schema-Dateien (`ap212.xsd`, `bond.xsd`, `petri.xsd`) wurden auf der Basis von XMI aus den UML/MOF-Ontologien, die im Abschnitt 5.1 beschrieben wurden, generiert ([Grose 2002]).

In `symmetry` werden die Baugruppen- und Verbindungsmorphismen, die im Kapitel 5.2 als OCL-Ausdrücke spezifiziert wurden, überprüft. Die Grundidee ist die, dass ausgehend von den Morphismen von einer gegebenen Produktstruktur auf Baugruppen- und Verknüpfungsrelationen geschlossen wird, die äquivalent in den jeweiligen Petrinetz- bzw. Bondgraphenmodellen vorliegen müssen. Ist dies nicht der Fall, so meldet `symmetry` einen Fehler unter Nennung der erwarteten Relationen. Es handelt sich also bei `symmetry` um ein Kritiksystem ([Puppe et al. 2003]), das Modelle überprüft und gegebenenfalls Vorschläge zur Modellierung notwendiger Relationen macht. Desweiteren untersucht `symmetry` die Abbildung der Produktmodell-Instanzen auf äquivalente Petrinetz- und Bondgraphen-Instanzen. Diese Abbildung beruht auf Namensgleichheit, so dass leicht

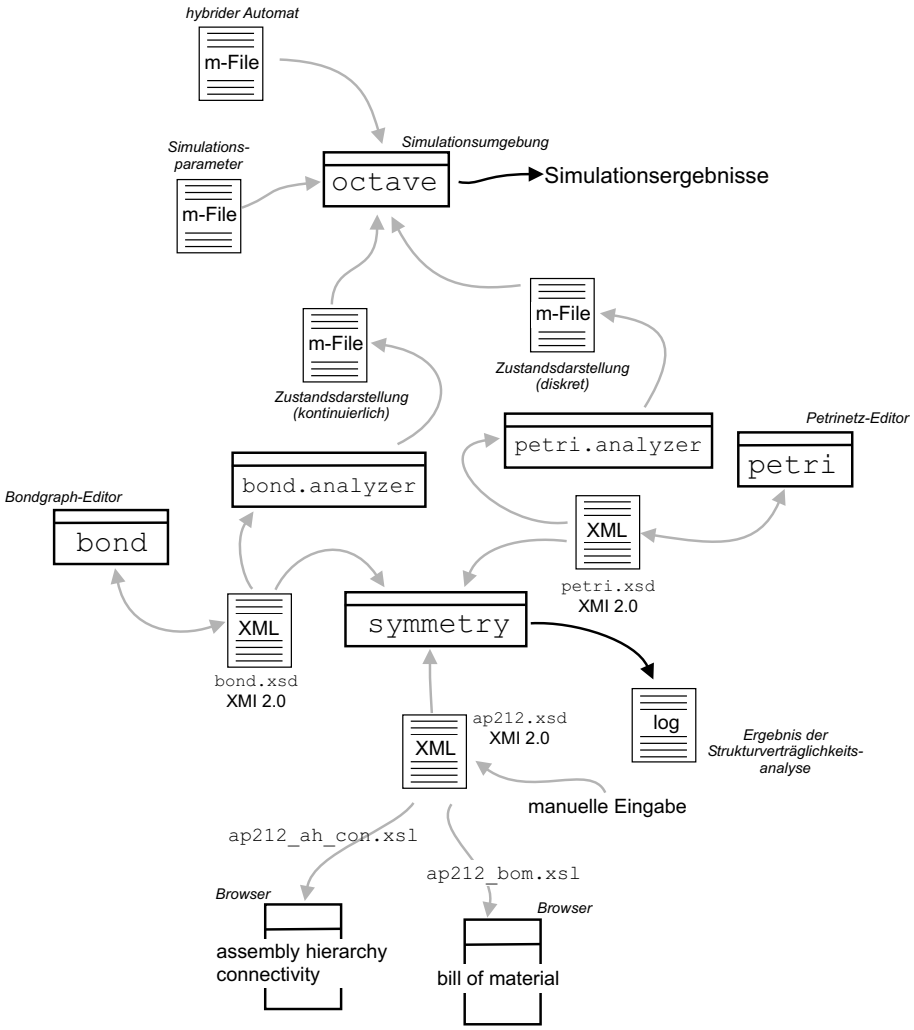


Abbildung 6.1: Werkzeugumgebung

überprüft werden kann, ob ein gegebenes Gerät aus dem Produktmodell regelgerecht in einem Bondgraphen oder Petrinetz modelliert wurde. Die Ergebnisse dieser Analysen werden in ein *log*-Datei geschrieben und danach abgespeichert.

Während die XML-Daten zum Produktmodell im momentanen Entwicklungsstand noch manuell eingegeben werden müssen, bietet die Werkzeugumgebung Java-basierte Editoren, die die Modellierung hierarchischer Bondgraphen (`bond`-Paket) und Petrinetze (`petri`-Paket) und deren Speicherung im XML-Format ermöglicht. Produktmodelldaten, die bereits im XML-Format vorliegen, können mit XSLT-Skripten mit gängigen Browsern übersichtlich eingesehen werden. Die XSLT-Skripte ermöglichen die Verarbeitung der XML-Produktdaten zu Baugruppenhierachien (`ap212_ah_con.xsl`) oder zu Stücklisten (`ap212_bom.xsl`).

Die Softwarepakete `bond.analyzer` und `petri.analyzer` bieten darüberhinaus die Möglichkeit, aus Petrinetz- und Bondgraphenmodellen, die als XML-Dateien vorliegen,

analysierbare Zustandsmodelle als *m-Files* zu generieren, siehe [Jiang 2005]. Die *m-Files* können dann von einer Simulationsumbegung wie Matlab oder Octave zu Simulationszwecken genutzt werden. Dies ist z.B. mit einem zur Übersetzungszeit bereits vordefinierten hybriden Automaten möglich, wie er in Gleichung (2.91) auf Seite 39 vorgestellt wurde. Desweiteren ist eine Parameterdatei notwendig, um alle für die Simulation notwendigen Konstanten und Kenngrößen vor der Simulation festlegen zu können.

Nachdem die Werkzeugumgebung kurz vorgestellt wurde, wird sie nun im folgenden beispielhaft zur Modellierung und Simulation eines 2-Tank-Systems genutzt, siehe [Nenninger 2002].

6.2 Beispiel: hybrides 2-Tank-System

Die folgende Abbildung 6.2 zeigt ein 2-Tank-System, bestehend aus zwei Tanks 1 und 2 und einer Pumpe, die den Tank 1 mit dem Durchfluss Q_0 befüllt . Tank 1 ist mit Tank 2 durch einen Überlauf, der sich auf der Tankhöhe h^* befindet, verbunden. Weiterhin sind beide Tanks durch ein schaltbares Absperrventil auf Bodenhöhe verbunden. Am Boden von Tank 2 ein Auslass, mit dem er entleert wird. In der Steuerung ist ein 2-Punkt-Regler implementiert, der auf Basis einer druckbasierten Füllstandsmessung in Tank 1 das Absperrventil ansteuert. Mit diesem Gerätebild kann man nun ein ISO 10303-

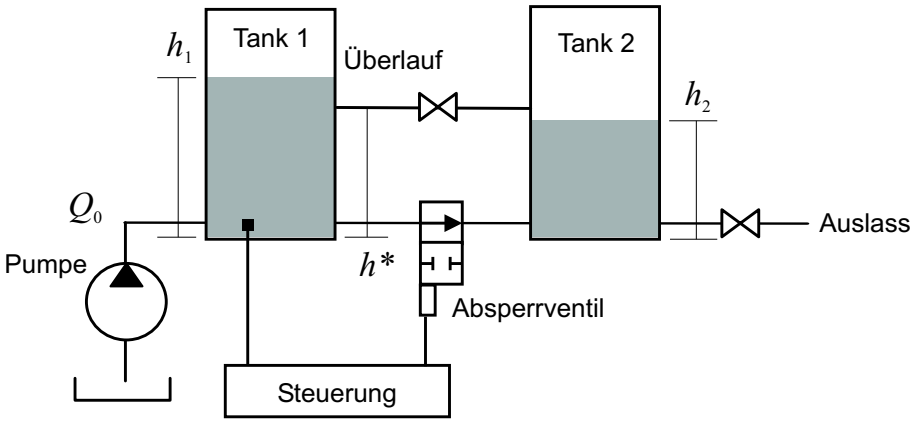


Abbildung 6.2: Hydraulisches Schaltbild eines 2-Tank-Systems

konformes Produktmodell in Form von AP212-Instanzen erstellen und als XML-Datei ablegen. Als Produktmodell dient das Klassendiagramm aus Abbildung 5.4.

Sämtliche Geräte und Anlagenteile sind Instanzen der Klasse single_device und werden zur Baugruppe '2-Tank-System' zusammengefasst. Diese Baugruppe ist eine Instanz der Klasse assmbly_definition und enthält neben dem Tank 1 und dem Tank 2 die Steuerung mit Sensor, die Pumpe und das Absperrventil. Des weiteren sind die verlustbehafteten Rohrleitungen am Auslass und zwischen Tank 1 und Tank 2 Teil dieser Baugruppe. Tank 1 und Tank 2 wiederum setzen sich aus dem eigentlichen Behälter und aus dem angeflanschten Überlauf zusammen.

Neben der Baugruppenhierarchie besitzt das 2-Tank-System auch eine Verschaltungsstrukur, die die Verknüpfung der Geräte untereinander bestimmt. Auf hydraulischer Seite sind dies die verlustfreien Verbindungsrohre und Leitungen. Auf steuerungsstechnischer Seite sind dies die elektrischen Leitungen zwischen der Steuerung, dem Füllstandssensor und dem Absperrventil. Die Verknüpfung der Geräte sind Instanzen der Klasse connection.

Electrotechnical System, ***ISO 10303, Part: ap212, Version: DIS***

assembly hierarchy

technical system: '2-Tank-System'

- [single device] '2-Tank-System'
  [assembly definition]
  - [single device] 'Tank 1'
    [assembly definition]
    - [single device] 'Behaelter 1'
    - [single device] 'Ueberlauf 1'
  - [single device] 'Pumpe'
  - [single device] 'Tank 2'
    [assembly definition]
    - [single device] 'Behaelter 2'
    - [single device] 'Ueberlauf 2'
  - [single device] 'Auslass'
  - [single device] 'Rohr'
  - [single device] 'Ventil'
  - [single device] 'Steuerung'
  - [single device] 'Fuellstand Sensor'

connectivity

- [connection] 'serial 3' connecting
  - 'Pumpe'
  - 'Tank 1'
- [connection] 'serial 4' connecting
  - 'Tank 2'
  - 'Auslass'
- [connection] 'serial 2' connecting
  - 'Tank 1'
  - 'Rohr'
  - 'Tank 2'
- [connection] 'serial 1' connecting
  - 'Tank 1'
  - 'Ventil'
  - 'Tank 2'
- [connection] 'auf' connecting
  - 'Ventil'
  - 'Steuerung'
- [connection] 'zu' connecting
  - 'Ventil'
  - 'Steuerung'
- [connection] 'fuellstand' connecting
  - 'Steuerung'
  - 'Fuellstand Sensor'

XSLT stylesheet, (C) iVA, TU Braunschweig
XSLT compiler Xalan, (C) open source, apache.org

Abbildung 6.3: Mit `ap212_ah_con.xsl` erzeugte Baugruppen- und Verschaltungsstruktur für das 2-Tank-System

Wie im unteren Teil von Abbildung 6.1 gezeigt, kann die XML-Datei des Produktmodells durch Umwandlung mittels XSLT benutzerfreundlich in einem Browser dargestellt werden. Die Abbildung 6.3 auf Seite 130 zeigt die Baugruppen- und Verschaltungsstruktur des 2-Tank-Systems nach Umwandlung der entsprechenden XML-Datei durch das `ap212_ah_con.xsl`-Skript mit dem Internet Explorer 6.0. Im oberen Teil *assembly hierarchy* finden sich die Geräte des 2-Tank-Systems in hierarchisch strukturierter Form. Der untere Teil *connectivity* zeigt die verlustfreien Verknüpfungen und Verschaltungen, die zwischen den einzelnen Geräten bestehen.

Mit einem weiteren XSLT-Skript (`ap212_bom.xsl`) kann das XML-Produktmodell zu einer Stückliste aufgearbeitet und angezeigt werden. Die Stückliste (*bill of material*) der im 2-Tank-System verbauten Teile ist in der Abbildung 6.4 dargestellt. Alle Baugruppen- und Verknüpfungsrelationen sind hier ausgeblendet. Anstelle dessen wird die Anzahl der verbauten Geräte und ihre Kategorie angezeigt. Als Gerätekategorie dient hier die Klasse design_discipline_item_definition (DDID) aus Abbildung 5.4.

Electrotechnical System, ***ISO 10303, Part: ap212, Version: DIS***

---

bill of material

technical system: '2-Tank-System'

Pos.	Part (DDID)	Amount
1	Pumpe	1
2	Rohr	2
3	Behaelter	2
4	Ueberlauf	2
5	Ventil	1
6	Steuerung	1
7	Sensor	1

XSLT stylesheet, (C) iVA, TU Braunschweig
XSLT compiler Xalan, (C) open source, apache.org

Abbildung 6.4: Mit `ap212_bom.xsl` erzeugte Stückliste

Neben der Pumpe und den beiden Behältern mit Überlauf findet man die Steuerungstechnik (Sensor, Steuerung, Ventil) und zwei Rohre, von denen das eine als Auslass und das andere als Verbindung zwischen Tank 1 und Tank 2 Verwendung findet.

Die Aufgabe besteht nun darin, das 2-Tanksystem mit der in Kapitel 6.1 vorgestellten Werkzeugumgebung mit Petrinetzen und Bondgraphen strukturtreu zu modellieren und anschließend zu simulieren.

6.2.1 Strukturtreue Modellbildung

Aufbauend auf dem hydraulischen Schaltbild wird zuerst ein hierarchischer Bondgraph entworfen, der die Struktur des 2-Tank-Systems wiedergibt. Der Bondgraph wird dann als XML-Datei abgelegt und durch das `symmetry`-Paket mit dem XML-Produktmodell aus Abbildung 6.3 verglichen. Die Abbildung 6.5 zeigt den Bondgraphen für das 2-Tank-System:

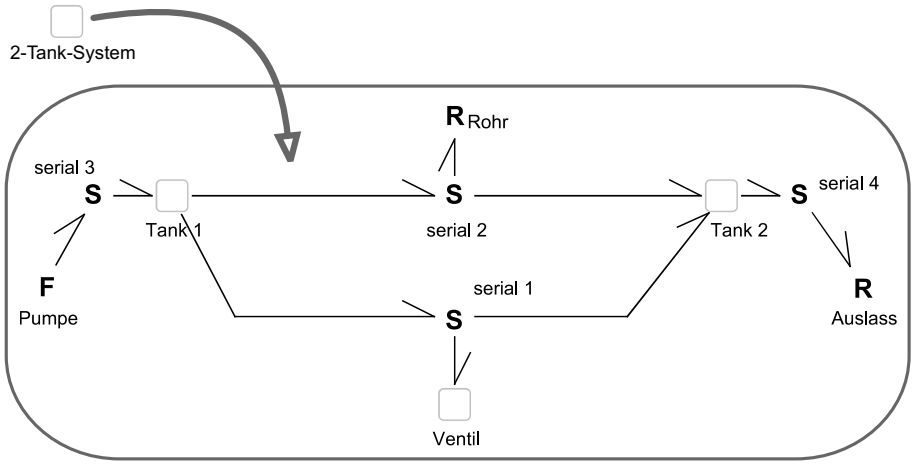


Abbildung 6.5: Bondgraph des 2-Tank-Systems

Auf der obersten Hierarchieebene findet sich ein einziger hierarchischer Knoten, der das gesamte 2-Tank-System repräsentiert und sich in einen weiteren Bondgraphen untergliedert. Pumpe, Rohr und Auslass sind hier als klassische 1-port-Knoten (*flow*-Quelle, Widerstand) modelliert. Tank 1, Tank 2 und das Ventil sind ebenfall hierarchische Knoten, die jeweils einen Subgraphen beinhalten.

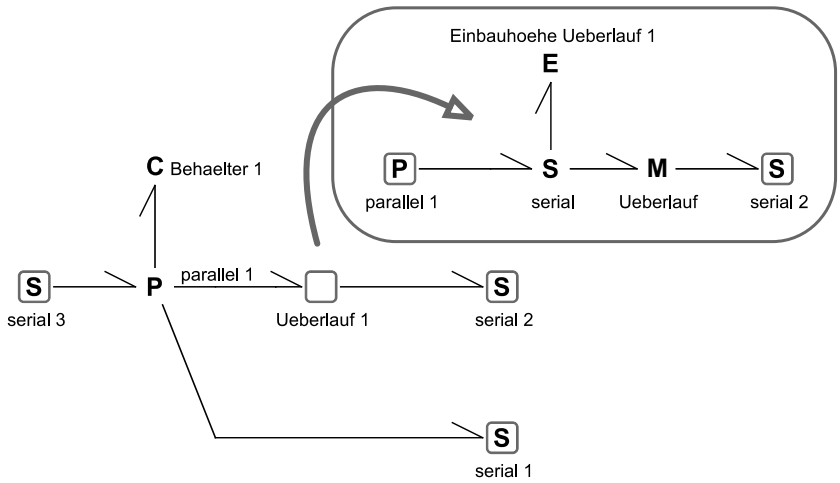


Abbildung 6.6: Bondgraph, Tank 1

Beispielhaft sei hier der Subgraph von Tank 1 erläutert (Abbildung 6.6). Tank 1 besteht aus der Paralleschaltung einer Kapazität, die das Speichervermögen des Behälters modelliert und dem Überlauf, der - wie aus dem rechten Teil von Abbildung 6.6 ersichtlich ist - aus der Serienschaltung eines modulierten Transformators mit einer Druckquelle besteht. Der Druck dieser Quelle ist der Einbauhöhe des Überlaufs proportional. Der Übertragungsfaktor des Transformators hängt vom Füllstand h_1 ab. Falls h_1 unter die Überlaufhöhe h^* sinkt ($p_1 - p^* < 0$), wird der Faktor zu 0 und der Druck am Überlauf 1 fällt auf den Umgebungsdruck $p_0 = 0$ ab. Im anderen Fall ist der Faktor 1, und es herrscht am Überlauf der Druck $p_1 - p^*$. Analog verhält sich der Überlauf von Tank 2.

Da sich Bondgraphen zur Modellierung solch diskreten Verhaltens nicht eignen, wird die Verhaltenslogik des Überlaufs mit einem Petrinetz spezifiziert (Abbildung 6.7):

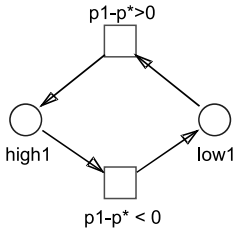


Abbildung 6.7: Petrinetz des Überlaufs

Je nach Füllstand ist entweder der Platz high1 oder der Platz low1 markiert. Die Umschaltung erfolgt, falls die Ausdrücke an den Transitionen wahr werden. Der Platz high1 korreliert mit dem Übertragungsfaktor des modulierten Transformators aus Abbildung 6.6, so dass der Überlauf abhängig von der Markierung dieses Netzes schaltet.

Bei dem Absperrventil handelt es sich ebenfalls um ein schaltbares Element, dessen Verhaltenslogik mit Petrinetzen spezifiziert wird. Auch hier dient ein modulierter Transformator zur Darstellung der Schaltvorgänge innerhalb des Bondgraphen. An den Transformator des Schaltventils ist lediglich ein Widerstandsknoten angeschlossen, mit dem der hydraulische Strömungswiderstand des Ventils modelliert ist. Der Bondgraph des Ventils sieht also sehr einfach aus und wird hier der Vollständigkeit wegen gezeigt (Abbildung 6.8).

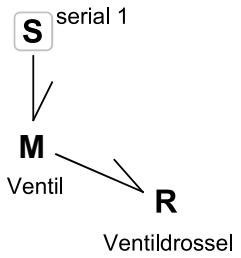


Abbildung 6.8: Bondgraph des Ventils

Den eben erläuterten Bondgraphen für das 2-Tank-System kann man nun mit dem Produktmodell durch das `symmetry`-Paket auf Strukturverträglichkeit überprüfen lassen. `symmetry` liest den Bondgraphen und das Produktmodell ein, erstellt aufgrund von Namensgleichheit Knoten/Geräte-Paare und überprüft, ob zwischen den Knoten im Bondgraphen die gleichen Baugruppen- und Verschaltungrelationen herrschen wie zwischen den namensgleichen Geräten. Das Ergebnis dieser Überprüfung zeigt das folgende Listing.

```
generated log file at 04.07.2004 02:46:25

--- MAPPING LOG ---
Match: single_device [_2_1] <--> iVA:Capacity  [_62]   'Behaelter 1'
Match: single_device [_2_2] <--> iVA:HiPort  [_96]   'Ueberlauf 1'
Match: single_device [_2_3] <--> iVA:Capacity  [_77]   'Behaelter 2'
Match: single_device [_2_4] <--> iVA:HiPort  [_111]   'Ueberlauf 2'
Match: single_device [_2_5] <--> iVA:HiPort  [_101]   'Tank 1'
Match: single_device [_2_6] <--> iVA:Flow  [_63]   'Pumpe'
Match: single_device [_2_7] <--> iVA:HiPort  [_116]   'Tank 2'
Match: single_device [_2_8] <--> iVA:Resistor  [_78]   'Auslass'
Match: single_device [_2_9] <--> iVA:Resistor  [_83]   'Rohr'
Match: single_device [_2_10] <--> iVA:HiPort  [_108]   'Ventil'
Error: no matching node found - node with name 'Steuerung' recommended
Error: no matching node found - node with name 'Fuellstand Sensor' recommended
Match: single_device [_0_2] <--> iVA:HiPort  [_123]   '2-Tank-System'

--- SYMMETRY LOG ---
   ASSEMBLY
Match: devices [_2_10, _0_2] <--> nodes [_108, _123]
Match: devices [_2_9, _0_2] <--> nodes [_83, _123]
Match: devices [_2_8, _0_2] <--> nodes [_78, _123]
Match: devices [_2_7, _0_2] <--> nodes [_116, _123]
Match: devices [_2_6, _0_2] <--> nodes [_63, _123]
Match: devices [_2_5, _0_2] <--> nodes [_101, _123]
Match: devices [_2_4, _2_7] <--> nodes [_111, _116]
Match: devices [_2_3, _2_7] <--> nodes [_77, _116]
Match: devices [_2_2, _2_5] <--> nodes [_96, _101]
Match: devices [_2_1, _2_5] <--> nodes [_62, _101]
   CONNECTIVITY
Match: single_device [_2_6, _2_5] <--> multiport [_63, _101]  'serial 3'
Match: single_device [_2_8, _2_7] <--> multiport [_78, _116]  'serial 4'
Match: single_device [_2_5, _2_9, _2_7] <--> multiport [_101, _83, _116]  'serial 2'
Match: single_device [_2_5, _2_10, _2_7] <--> multiport [_101, _108, _116]  'serial 1'
Error: no port connecting [null, _108] with name 'auf' found!
Error: no port connecting [null, _108] with name 'zu' found!
Error: no port connecting [null, null] with name 'fuellstand' found!
```

Im ersten Teil (`MAPPING LOG`) wird die Übereinstimmung von Geräte/Knoten-Paaren ausgegeben. Außer den steuerungstechnischen Geräten (Steuerung und Füllstandsensor) wurden alle Systemkomponenten im Bondgraphen modelliert. Da die Steuerung und der

Füllstandsensor Signale geringer Energie verarbeiten, ist es nicht sinnvoll, diese in die energetische Modellbildung mit dem Bondgraphen aufzunehmen. Vielmehr werden diese Komponenten im Petrinetz berücksichtigt.

Der zweite Teil (`SYMMETRY LOG`) der Ausgabe untergliedert sich in die Überprüfung der Baugruppen- (`ASSEMBLY`) und der Verschaltungrelationen (`CONNECTIVITY`). Während sämtliche Bondgraphenknoten in einer zum Produktmodell analogen Hierarchie angeordnet wurden, sind in der Überprüfung der Verschaltungen Fehler zu verzeichnen: die Verknüpfungen zwischen der Steuerung, dem Füllstandsensor und dem Absperrventil sind im Bondgraphen nicht zu finden. Dies liegt einfach daran, dass sowohl der Füllstandsensor als auch die Steuerung im Bondgraphenmodell fehlen und deshalb auch die entsprechenden Verknüpfungen fehlen müssen. Aber abgesehen von den nur steuerungstechnisch relevanten Komponenten (Steuerung, Sensor und Signalleitungen) wurde das Produktmodell des 2-Tank-Systems strukturtreu in einen Bondgraphen umgesetzt.

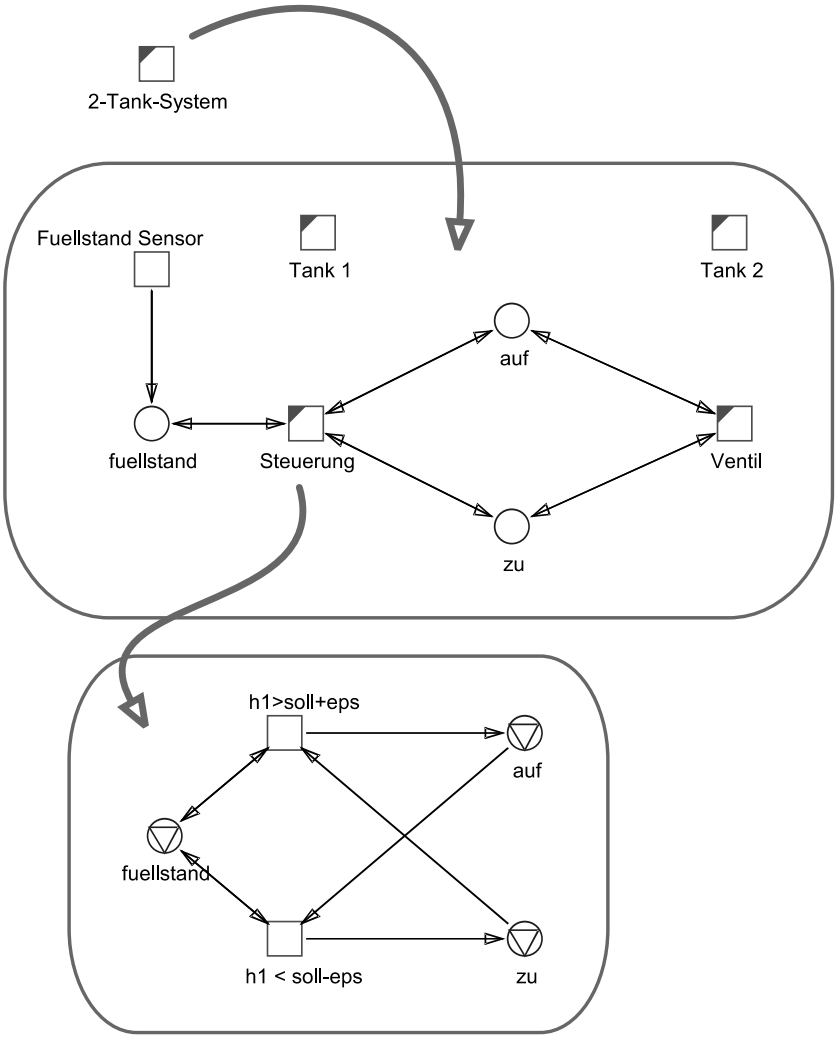


Abbildung 6.9: Petrinetz des 2-Tank-Systems

Während die Energieumsetzungsphänomene innerhalb des 2-Tank-Systems mit Bondgraphen modelliert werden, dienen Petrinetze zur Modellierung der Informations- bzw. Kommunikationsphänomene. Neben dem Steuerprogramm beinhaltet dies auch die füllstandsabhängigen Überlaufzustände der Strecke. Im folgenden wird nun die Modellbildung des 2-Tank-Systems mit Petrinetzen vorgestellt und anschließend überprüft, inwieweit die Modellbildung strukturtreu ist.

Die Abbildung 6.9 zeigt das Petrinetzmodell für das 2-Tank-System. Ähnlich wie beim Bondgraphenmodell in Abbildung 6.5 bildet die verfeinerte Transition 2-Tank-System die oberste Hierarchiebene des Petrinetzes. Das unterlagerte Netz enthält weitere verfeinerte Transitionen, mit denen Tank 1 und Tank 2, das Ventil und die Steuerung modelliert sind. Der Füllstandssensor wird durch eine einfache Transition repräsentiert und ist über eine einfache Kante mit dem Füllstandsplatz verbunden, so dass dieser im Verlauf der Simulation immer markiert ist. Es liegt also immer ein Füllstand vor. Dieses triviale Verhalten muss weiter verfeinert werden, wenn z.B. zuverlässigkeitsrelevante Phänomene wie z.B. Ausfall und Wartung modelliert werden sollen.

Da die Steuerung auf die Füllstandsdaten zugreift, ist die Steuerung ebenfalls mit dem Füllstandsplatz verbunden. Die Steuerung dient der Aufrechterhaltung eines Sollfüllstands im Tank 1. Ihr liegt ein 2-Punkt-Regler zugrunde, dessen Petrinetz im unteren Teil von Abbildung 6.9 dargestellt ist. Die Umschaltung der Plätze auf und zu wird durch die mit arithmetischen Ausdrücken versehenen Transitionen realisiert. Über diese Plätze, die die entsprechenden Signalleitungen in der Gerätewelt strukturtreu abbilden, kommuniziert die Steuerung mit dem Absperrventil.

Die Transition Tank 1 enhält nur das bereits in Abbildung 6.7 gezeigte einfache Petrinetz des Überlaufs. Ähnliche Netze sind auch in den Transitionen Tank 2 und Ventil enthalten, um die dortigen Betriebsumschaltungen (hoher Füllstand/niedriger Füllstand bzw. auf/zu) zu modellieren. Andere Informations- oder Kommunikationsphänomene werden nicht modelliert.

Die Überprüfung des Petrinetzmodells auf Strukturtreue kann ebenfalls mit dem `symmetry`-Paket durchführt werden. Das Ergebnis dieser Überprüfung zeigt das folgende Listing:

```
generated log file at 04.07.2004 02:51:24

--- MAPPING LOG ---
Error: no matching transition found -  transition with name 'Behaelter 1' recommended
Match:single_device [_2_2] <--> Transition [_50]   'Ueberlauf 1'
Error: no matching transition found -  transition with name 'Behaelter 2' recommended
Match:single_device [_2_4] <--> Transition [_62]   'Ueberlauf 2'
Match:single_device [_2_5] <--> Transition [_52]   'Tank 1'
Error: no matching transition found -  transition with name 'Pumpe' recommended
Match:single_device [_2_7] <--> Transition [_64]   'Tank 2'
Error: no matching transition found -  transition with name 'Auslass' recommended
Error: no matching transition found -  transition with name 'Rohr' recommended
Match:single_device [_2_10] <--> Transition [_24]  'Ventil'
Match:single_device [_2_20] <--> Transition [_30]  'Steuerung'
Match:single_device [_2_21] <--> Transition [_13]  'Fuellstand Sensor'
```

```
Match:single_device [_0_2] <--> Transition [_66]   '2-Tank-System'

--- SYMMETRY LOG ---
   ASSEMBLY
Match:single_device [_2_10, _0_2] <--> Transition [_24, _66]
Match:single_device [_2_7, _0_2] <--> Transition [_64, _66]
Match:single_device [_2_21, _0_2] <--> Transition [_13, _66]
Match:single_device [_2_20, _0_2] <--> Transition [_30, _66]
Match:single_device [_2_5, _0_2] <--> Transition [_52, _66]
Match:single_device [_2_4, _2_7] <--> Transition [_62, _64]
Match:single_device [_2_2, _2_5] <--> Transition [_50, _52]
   CONNECTIVITY
Error: no place connecting [null, _52] with inscription 'serial 3' found!
Error: no place connecting [null, _64] with inscription 'serial 4' found!
Error: no place connecting [_52, null, _64] with inscription 'serial 2' found!
Error: no place connecting [_52, _24, _64] with inscription 'serial 1' found!
Match: single_device [_2_20, _2_10] <--> Transition [_30, _24] 'auf'
Match: single_device [_2_20, _2_10] <--> Transition [_30, _24] 'zu'
Match: single_device [_2_20, _2_21] <--> Transition [_30, _13] 'fuellstand'
```

Am Anfang der Ausgabe werden die Übereinstimmungen zwischen Geräten und Transitionen registriert (`MAPPING LOG`). Da im Petrinetz nicht alle Geräte wie z.B. die Pumpe oder der Auslass sinnvoll nachmodelliert werden konnten, ergeben sich hier erwartungsgemäß Fehlermeldungen. Dem `SYMMETRY LOG` ist jedoch zu entnehmen, dass die vorhandenen Transitionen in einer der Gerätewelt entsprechenden Hierarchie angeordnet sind. Die Tatsache, dass nicht alle Geräte modelliert wurden, wirkt sich bei der Überprüfung der Baugruppenbeziehungen nicht aus, so dass hier keine Fehlermeldungen zu verzeichnen sind. Im Gegensatz dazu ergeben sich jedoch Fehler bei der Überprüfung der Verschaltungsbeziehungen. Sämtliche Fehler beziehen sich hier auf die im Petrinetz nicht vorhandene Modellierung der verlustbehafteten hydraulischen Leitungen und Rohre (`serial 1` bis `serial 4`). Da diese für die Modellbildung der informationsverarbeitenden Prozesse nicht relevant sind, ist der Verzicht auf entsprechende Kanäle im Petrinetz gerechtfertigt.

Die Signalverknüpfungen zwischen der Steuerung, dem Sensor und dem Absperrventil, die ja im Produktmodell vorhanden sind, jedoch im Bondgraphen fehlen, wurden nun im Petrinetz durch entsprechende Plätze korrekt modelliert. Ähnliches gilt auch für die hydraulischen Komponenten, die im Petrinetz fehlen und im Bondgraphen vorhanden sind. Unter diesen Gesichtspunkten kann also festgestellt werden, dass das 2-Tank-System mit einer kombinierten Petrinetz/Bondgraphenbeschreibung strukturtreu nachgebildet wurden, so dass alle Geräte und die zwischen diesen Geräten herrschenden Baugruppen- und Verschaltungsbeziehungen korrekt modelliert wurden.

6.2.2 Simulationsergebnis

Nachdem im vorigen Kapitel 6.2.1 gezeigt wurde, wie das 2-Tank-System mit kombinierten Petrinetz/Bondgraphen-Modellen strukturtreu beschrieben werden kann, werden

diese Modelle im folgenden nun zur Simulation verwendet. Dazu werden mit den Paketen `bond.analyzer` bzw. `petri.analyzer` die als XML-Dateien vorliegenden Petrinetz- und Bondgraphenmodelle gemäß Abschnitt 2.5 in diskrete und kontinuierliche Zustandsmodelle überführt und als m-Files (`c_state.m` und `d_state.m`) exportiert. Diese Zustandsmodelle werden dann mit einem ebenfalls als m-File vorliegenden hybriden Automaten simuliert (`solver.m`), der unabhängig von den aktuell generierten Zustandsmodellen ist und auf der Gleichung 2.91 auf Seite 39 basiert. Zur Lösung des hybriden Automaten wird ein einfaches Euler-Verfahren verwendet, bei dem zuerst die Gleichungen des diskreten Teilautomaten gelöst wird und dabei auf die quantisierten kontinuierlichen Zustandsgrößen des vorigen Zeitschritts zurückgriffen wird. Danach können dann bei der Lösung des kontinuierlichen Teilautomaten die aktuellen diskreten Zustandsgrößen genutzt werden, ohne dass es zu algebraischen Schleifen kommt (siehe hierzu Abschnitt 2.4.2.2).

Die notwendigen Simulationsparameter sind ebenfalls in einem m-File abgelegt und müssen vor jeder Simulation geladen werden (`parameters.m`). Als eigentliche Simulationsumgebung dient die numerische Berechnungssoftware `octave`, [Eaton 1997].

Die folgende Abbildung 6.10 gibt nun zusammenfassend einen Überblick über die bei der Simulation verwendeten Dateien und ihre funktionale Verkopplung.

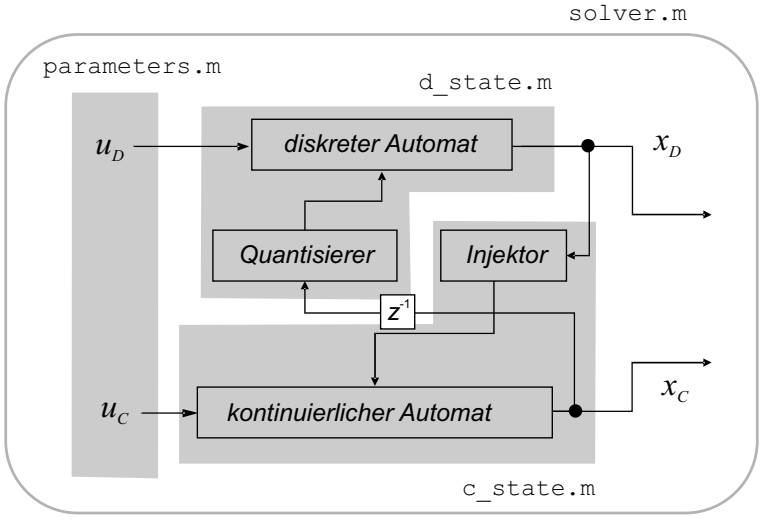


Abbildung 6.10: Die im Verlauf der Simulation verwendeten bzw. erzeugten m-Files

Mit den in Abbildung 6.10 gezeigten Dateien kann nun eine Simulation durchgeführt werden. Als Anfangsbedingungen besitzen die Tanks die normierten Füllhöhen von 12 und 4. Die Sollhöhe des Tanks 1 ist 8, der Überlauf ist waageracht auf der normierten Höhe 9 montiert.

Die folgenden Abbildungen 6.11 und 6.12 zeigen die zeitlichen Verläufe der Füllstände und die Schaltzustände des Absperrventils und des linken Überlaufs 1. Da die Anfangsfüllhöhe des linken Tanks die Höhe des Überlaufs überschreitet, leitet der Überlauf die

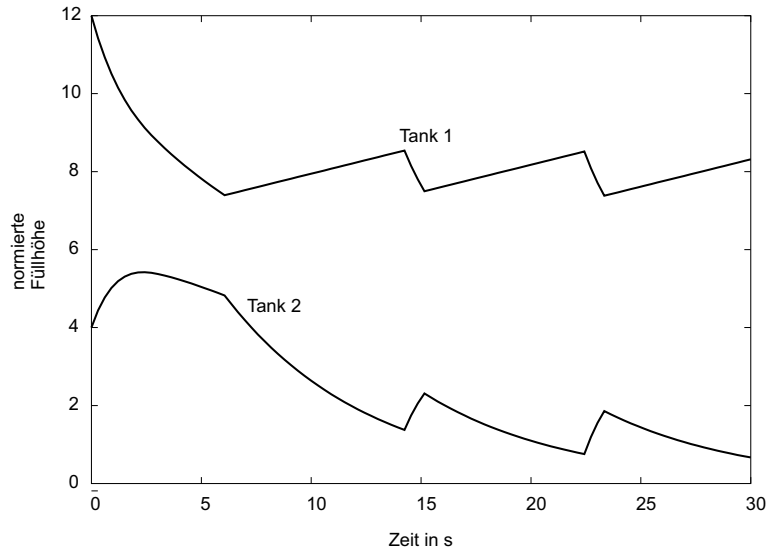


Abbildung 6.11: Zeitlicher Verlauf der Füllhöhe in Tank 1 und Tank 2

Flüssigkeit vom linken in den rechten Tank ab. Nach ca. 3 Sekunden sinkt der Füllstand unter die Höhe des Überlaufs. Kurz danach fällt der Füllstand unter die Sollhöhe, und die Steuerung schließt das Absperrventil. Die Füllhöhe im Tank 1 steigt wieder, bis das Absperrventil erneut öffnet, so dass der Grenzzyklus der 2-Punkt-Regelung beginnt.

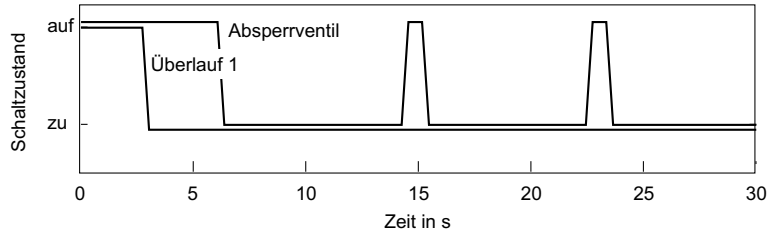


Abbildung 6.12: Schaltzustand des Absperrventils und des Überlaufs 1

6.3 Zusammenfassung

Gegenstand dieses Kapitels ist die Vorstellung einer Werkzeugumgebung, die die strukturtreue Modellbildung mit Petrinetzen und Bondgraphen und die anschließende Simulation ermöglicht. Die strukturtreue Modellbildung basiert auf der Überprüfung der Baugruppen- und Verknüpfungsmorphsimen, die zwischen Bondgraphen bzw. Petrinetzen auf der einen Seite und dem Produktmodell nach ISO 10303 auf der anderen Seite herrschen. Diese Morphismen wurden im vorherigen Kapitel 5 eingehend vorgestellt.

Im Zentrum dieser Werkzeugumgebung steht ein unter Java programmiertes Software-Paket (`symmetry`), das Petrinetz- und Bondgraphenmodelle als XML-Dateien einliest

und diese mit einer gegebenen Produktstruktur, die auf dem Produktmodell von ISO 10303, AP 212 basiert und ebenfalls im XML-Format vorliegt, vergleicht.

Während die XML-Dateien zum Produktmodell im momentanen Entwicklungsstand manuell eingegeben werden müssen, bietet die Werkzeugumgebung Java-Editoren (`bond`- und `petri`-Paket), die die Modelleriung hierarchischer Bondgraphen und Petrinetze und deren Speicherung im XML-Format ermöglichen. Die XML-Dateien für die Produktmodelldaten können jedoch mit XSLT-Skripten zu Baugruppenhierarchien oder Stücklisten verarbeitet und mit gängigen Web-Browsern übersichtlich dargestellt werden.

Zusätzlich hat man mit den Softwarepaketen `bond.analyzer` und `petri.analyzer` die Möglichkeit, aus den Petrinetz- und Bondgraphenmodellen simulierbare und analysierbare Zustandsmodelle als *m-Files* zu generieren, die von einer numerischen Berechnungssoftware wie Matlab oder Octave zu Simulationszwecken genutzt werden können.

Als Beispiel wurde ein 2-Tank-System mit Bondgraphen und Petrinetzen strukturtreu modelliert und anschließend simuliert, um so die Durchgängigkeit der zur Verfügung gestellten Werkzeugkette zu überprüfen.

7 Zusammenfassung und Ausblick

Um einen integrierten Entwurfsprozess von Automatisierungssystemen gewährleisten zu können, ist idealerweise die Berücksichtigung der unterschiedlichen Sichten zur Beschreibung des Aufbaus, des Verhaltens, der Funktion und des Zustands eines Systems notwendig. Wie gezeigt wurde bietet die Produktdatentechnologie nach ISO 10303 (STEP - *Standard for the Exchange of Product Model Data*) mit seinen umfangreichen produktdefinierenden Referenzmodellen für die Beschreibung von Gerätestrukturen (Aufbausicht) hierfür eine gute Integrationsgrundlage.

Desweiteren sind bei der notwendigen Verhaltensbeschreibung die prozesstheoretischen Eigenheiten automatisierungstechnischer Systeme zu berücksichtigen. Hier ist insbesondere die kooperative Wechselwirkung in hybriden Systemen zu erwähnen. Bei dem in dieser Arbeit vorgestellten Entwurfsansatz hat sich deshalb ein Konzept zur kombinierten Verhaltensbeschreibung mit schaltenden Bondgraphen und mit Stellen/Transitions-Netzen bewährt.

Diese Beschreibungsmittel wurden ausführlich erläutert und mit Techniken der *Object Modeling Architecture* (OMA) in Ontologien überführt. Auf diese Ontologien konnte mittels strukturverträglicher Abbildungen (Morphismen) eine STEP-konforme produktdefinierende Ontologie, die auf dem Anwendungsprotokoll AP 212 für elektrotechnische Systeme beruht, abgebildet werden. Das Ergebnis dieses Konzeptes zur Integration strukturverträglicher Ontologien ist ein Verfahren für die strukturerhaltende Modellbildung, mit der vorhandene Produktstrukturen in der Verhaltensbeschreibung konsistent nachgeführt werden können, um somit Informationen aus der Aufbausicht angemessen in der Verhaltenssicht berücksichtigen zu können.

Die erarbeiteten Morphismen verhalten sich strukturverträglich bezüglich Baugruppen- und Verknüpfungsrelationen. Dementsprechend wurden die eingesetzten Bondgraphen und Petrinetze um ein Hierarchisierungskonzept erweitert, das auf der Verhaltensseite beliebige Baugruppenhierarchien abbilden kann. Desweiteren wurden bestehende Analyseverfahren aufgegriffen und angepasst, um die bondgraphen- und petrinetzbasierten Verhaltensmodelle in eine kombinierte Zustandsdarstellung überführen zu können.

Das vorgestellte Verfahren zum integrierten Systementwurf wurde prototypisch implementiert. Diese Implementierung besteht aus Java-basierten Anwendungen für die Modellbildung, die Strukturverträglichkeitüberprüfung und die Systemanalyse und verwendet die Berechnungssoftware `octave` für die Simulation und XML-Technologien zum Datentransfer und zur Modelldatenrepräsentation. Die Validation erfolgte anhand eines einfachen 2-Tank-Systems mit Füllstandsregelung.

Im Laufe der Entstehungszeit dieser Dissertation hat sich ein Fülle von Möglichkeiten ergeben, wie das vorgestellte Entwurfsverfahren weiterentwickelt bzw. auf andere Modellkonzepte und Systemsichten ausgeweitetet werden könnte.

Zunächst müsste auf der Strukturseite die produktdefinierende Ontologie auf die generischen Ressourcen von ISO 10303 portiert werden, um so unabhängig vom verwendenten Anwendungsprotokoll, das ja die Terminologie elektrotechnischer Systeme verwendet, einen weiten Bereich branchenneutraler Produktdaten beim Entwurfsprozess verarbeiten zu können. Dies wäre auch notwendig, um Datenbestände aus gängigen PDM-Systemen nutzen zu können.

Auf der Ebene der Verhaltensbeschreibung sind erst jüngst objektorientierte Modellierungsmethoden und Beschreibungsmittel (z.B. Modelica, vgl. hierzu [Merz 2000]) insbesondere im akademischen Bereich immer weiter auf dem Vormarsch - über 40 Jahre nach ihrer revolutionären Entstehung in Form von SIMULA! Auch objektorientierte Verhaltensmodelle ließen sich mit Produktdaten durch Morphismen strukturtreu verkoppeln, da diese ebenso wie die hier gewählten Bondgraphen auf der nicht-kausalen Prozesskooperation beruhen. Die Übertragung in Zustandsform muss jedoch aufgrund der vielen Modellbildungsfreiheiten bei objektorientierten Methoden im Einzelfall geprüft werden.

Die funktionalen Aspekte des Systementwurfs wurde in dem vorgestellten Verfahren bisher nur ansatzweise berücksichtigt (Funktionsspezifikation eines 2-Punkt-Reglers mit Petrinetzen). Hier müsste näher untersucht werden, inwieweit typische Beschreibungsmittel aus dem Bereich der Funktionsspezifikation wie z.B. Funktionsblöcke in die benachbarten Systemsichten integriert werden können.

Dies gilt ebenso für Spezifikationstechniken für Zuverlässigkeits- und Sicherheitsuntersuchungen. Wie bereits erwähnt könnten diese Spezifikationen im Zustandsraum durchgeführt und dazu genutzt werden, die Verhaltensbeschreibung gegebener Gerätestrukturen auf Einhaltung von Sicherheitsanforderungen zu untersuchen.

Abschließend sei darauf verwiesen, dass das vorgestellte Verfahren bisher teilautomatisiert abläuft und nur die Systemanalyse unterstützt, da ausgehend von Produktmodellen strukturtreue Verhaltensmodelle gebildet werden können. In einem ersten Schritt könnte das Verfahren dadurch automatisiert werden, dass jedes Produkt mit einem bestimmten Modellbildungselement z.B. in Form von Datenbanken verknüpft wird, um beliebige Gerätekonfigurationen strukturverträglich und rechnergestützt in Verhaltensmodelle zu überführen. Der zweite Schritt führt dann von der Systemanalyse zur Systemsynthese, um aus Verhaltensmodellen passende Gerätebeschreibungen zu generieren. Wie man aber leicht sieht, sind die notwendigen Geräte/Modellbildungselemente-Verknüpfungen für domänenunabhängige Beschreibungsmittel wie Bondgraphen und Petrinetze nicht-injektive Abbildungen, so dass die automatisierte Systemsynthese schon aus diesen grundlegenden Überlegungen heraus das weitaus schwerer implementierbare Entwurfsverfahren darstellt.

A Abkürzungen

AIM	Application Interpreted Model
AP	Application Protocol
ARM	Application Reference Model
B/E-Netz	Bedingungs/Ereignis-Netz
DAML	DARPA Agents Markup Language
DARPA	Defense Advanced Reserach Projects Agency
DTD	Document Type Definition
EBNF	Erweiterte Backus/Naur Normalform
GMA	VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik
ISO	International Standards Organisation
K/I-Netz	Kanal/Instanzen-Netz
KIF	Knowledge Interchange Format
MKS	Mehrkörper-Simulation
MOF	Meta Object Facility
OADF	Object Analysis & Design Facility
OCL	Object Constraint Language
OIL	Ontology Inference Layer
OMA	Object Modeling Architecture
OMG	Object Management Group
OMT	Object Modeling Technique
OOD	Object Oriented Design
OOSE	Object Oriented Software Engineering
OWL	Ontology Web Language
P/T-Netz	Prädikat/Transitions-Netz
PDM	Product Data Management
PLM	Product Lifecycle Management
RDF	Resource Description Framework
S/T-Netz	Stellen/Transitions-Netz
SIMULA	Simulation Language
SPS	Speicherprogrammierbare Steuerung
STEP	Standard for the Exchange of Product Model Data
UML	Unified Modeling Language
UoF	Unit of Functionality
XMI	XML Metadata Interchange
XML	eXtensible Markup Language
XSLT	XML Stylesheet Language Transformation

B Glossar

Aufbau materiale und räumliche Struktur einer technischen Einrichtung. Dies beinhaltet alle Geräte und Bauelemente, deren Konfiguration in Baugruppen, alle physikalischen Verknüpfungen und Netzwerke zwischen den Baugruppen und Geräten und die topologische Anordnung und Orientierung (S. 1).

Bondgraph Beschreibungsmittel zur Darstellung von Phänomenen der Energieumsetzung (Speicherung, Dissipation, Umwandlung, Verteilung). Mit ihnen können z.B. Systeme aus den Bereichen Mechanik, Elektrizität und Hydraulik einheitlich modelliert, analysiert (→Kausalität im Sinne der Kooperation) und in eine Zustandsdarstellung (→Markov-Prozess) überführt werden (S. 86).

determinierter Prozess bei einem determinierten Prozess haben alle Signale mit gleicher Vergangenheit denselben zukünftigen Verlauf. Somit ist deren Zukunft eindeutig bestimmt (S. 21, Gl. (2.42)).

diskreter Prozess Prozess mit abzählbaren und geordneten oder ungeordneten Trägermengen (→Signal), der irreversibel und nicht-determiniert sein kann (S. 38).

Eigenschaft Merkmal eines Objektes mit →Kategorie und Wert (Ausprägung).

Funktion (Mathematik) Zuordnungsvorschrift, die einer Größe x eine zweite Größe y eindeutig zuordnet; im Sinne der Algebra eine linkstotale, rechteindeutige binäre Relation [Reinhardt 1991].

Funktion (Technik) Aufgabe oder Zweck, den eine technische Einrichtung erfüllen soll (S. 1).

Gedächtnis bestimmte Dauer oder Zeitspanne in der Vergangenheit eines →Signals, die für dessen weiteren Verlauf prägend ist (S. 19). Verlaufen zwei Signale innerhalb ihres gesamten Gedächtnisbereichs identisch, sind sie miteinander kompatibel und können damit im weiteren Prozessverlauf durch den zukünftigen Verlauf des jeweils anderen Signals fortgesetzt werden. Prozesse werden auf der Basis ihres Gedächtnisbereichs in verschiedene Prozessklassen unterschieden (→Markov-Prozesse, Differenzialprozesse usw.)

hybrider Prozess ein kooperativer Prozess, bei dem →diskrete und →kontinuierliche Teilprozesse miteinander in →kooperativer Wechselwirkung stehen (S. 36).

Informationsmodell die mit den Verfahren der Wissensverbeitung generierten formalen Modelle, die zur Strukturierung bzw. Repräsentation von deklarativem Anwendungswissen dienen.

Kategorie Name einer Objekteigenschaft.

Kausalität (im Sinne der Kooperation) definierte Berechnungsreihenfolge in mehrdimensionalen kooperativen Prozessen, die auf der Unterteilung der Signale eines Prozesses in Eingangs- und Ausgangsgrößen beruht. Bei bestimmten Beschreibungsmitteln wie z.B. →Bondgraphen liegt die Kausalität zur Zeit der Modelbildung nicht fest und kann auf der Basis von Analysemethoden bestimmt werden (S. 30).

Kausalität (im Sinne der Zeit) Prinzip, das besagt, dass zukünftige Ereignisse keinen Einfluss auf die Prozessvergangenheit ausüben können. Dieses Prinzip wird auch Temporalprinzip oder Kausalprinzip genannt (S. 29).

Klasse elementares Strukturierungskonzept der Objektorientierung. Ergebnis eines klassenbildenden Abstraktionsprozesses, bei dem →Objekte zu einer Klasse zusammengefasst werden, wenn sie hinsichtlich einer →Eigenschaft mindestens eine gemeinsame →Kategorie besitzen. Klassen sind Ausprägungsmuster für Objekte, die man Instanzen einer Klasse nennt (S. 66).

kontinuierlicher Prozess Prozess mit abzählbaren oder überabzählbaren, geordneten Trägermengen (→Signal), der immer →reversibel und →determiniert (funktional) ist (S. 38).

Konzeptualisierung Grundlage für die Spezifikation von deklarativem Wissen bezüglich eines gegebenen Gegenstandsbereiches mit den formalen Mitteln der mathematischen Logik. Eine Konzeptualisierung besteht aus einer Menge D von →Objekten aus dem Gegenstandsbereich und einer Menge R von n-stelligen Relationen über D (S. 43, Gl. (3.1)). Konzeptualisierungen sind sprach- und fallabhängig und können deshalb i.A. nur näherungsweise bestimmt werden.

Markov-Prozess →Prozesse ohne →Gedächtnis (S. 20, Gl. (2.38)). Mit genauer Kenntnis des Ist-Zustandes lässt sich die Prozesszukunft bestimmen. Kenntnisse über die Prozessvergangenheit sind nicht weiter notwendig. Die Beschreibung von Markov-Prozessen auf der Basis von Vor- und Nachzuständen (Transformationsprinzip) wird Zustandsdarstellung genannt (S. 32, Gl. (2.77)).

Morphismus Abbildung h von Menge A auf Menge B, bei der bestimmte Relationen μ über A und λ über B nach erfolgter Transformation durch h gleichermaßen gelten: $\mu(a_1, \ldots, a_n) \Rightarrow \lambda(b_1, \ldots, b_n) = \lambda(h(a_1), \ldots, h(a_n))$ (S. 57, Gl. (3.30). Der Morphismus h wird auch relationentreue, strukturtreue oder strukturverträgliche Abbildung genannt.

Objekt Gegenstände der Wahrnehmung, die je nach Anwendungsfall durch bestimmte →Eigenschaften (Attribute) beschrieben werden können.

Objektorientierung Methode zur Informationsmodellierung und Systemanalyse, die auf den Architekturprinzipien 'Zerlegung' und 'Abstraktion' beruht. Wichtige Konzepte der Objektorientierung sind Objektidentität, Klassenbildung und Vererbung, Attributierung und Aggregation. Objektorientierte Programmiersprachen zeichnen sich desweiteren durch Polymorphie und dynamisches Binden aus (S. 65).

Ontologie (Philosophie) Allgemeine Metaphysik, die sich mit der Grundstruktur des Seienden (Wirkliches und Nichtwirkliches) beschäftigt ([Meixner 2004]).

Ontologie (Informatik) ein in einer logischen Sprache formuliertes System von Aussagen zur Beschreibung von deklarativem Wissen über einen Gegenstandsbereich. Mit Ontologien kann eine →Konzeptualisierung nur näherungsweise approximiert werden (S. 47 und S. 49, Gl. (3.17)). Neben der Konzeptualisierung sind →Taxonomien, Vererbung und Instanzierung wichtige Konzepte zur Bildung von Ontologien.

Petrinetz Beschreibungsmittel zur Darstellung organisatorischer Vorgänge beliebiger Art [Petri 1962], mit der →Markov-Prozesse spezifiziert werden können. Sie kommen z.B. in der Informatik und Automatisierungstechnik zur Beschreibung des →Verhaltens informationsverarbeitender Prozesse zum Einsatz und bieten vielfältige Analyseverfahren wie z.B. Lebendigkeit und Erreichbarkeit (S. 79).

Produktmodell zuweilen sehr umfangreiches Informationsmodell zur Beschreibung der Daten, die während des gesamten Lebenszyklus' (Entwicklung, Herstellung, Betrieb, Ausmusterung) eines Produktes relevant sind. Beispiel hierfür sind die integrierten, generischen Ressourcen und die Anwendungsprotokolle der Normenreihe ISO 10303/STEP (S. 93).

Prozess Menge von →Signalen, die den zeitlichen Verlauf der Eigenschaftswerte eines Prozessobjektes beschreiben (S. 8, Gl. (2.5)). Eine wichtige Klasse von Prozessen sind die →Markov-Prozesse, bei denen ein Vor- in einen Nachzustand transformiert wird (Transformationsprinzip).

reversibler Prozess bei einem reversiblen Prozess haben alle Signale mit gleicher Zukunft denselben vergangenen Verlauf hinter sich. Somit ist deren Vergangenheit eindeutig bestimmt (S. 21, Gl. (2.43)).

Signal Mathematische →Funktion, die einen Zeitbereich T auf einen Phasenraum X abbildet und dadurch den zeitlichen Verlauf von Prozesseigenschaften charakterisiert. Die Mengen T und X heißen Grund- oder auch Trägermengen eines →Prozesses (S. 7, Gl. (2.1)).

Strukturverträglichkeit bezeichnet die Existenz von →Morphismen zwischen zwei algebraisch strukturierten Mengen M_1 und M_2. Im Zusammenhang von Ontologien

ist die Strukturverträglichkeit ein Kriterium zur Integration von Ontologien unterschiedlicher Gegenstandsbereiche und Sprachen (S. 56 und S. 59, Gl. (3.31)).

Taxonomie grundlegendes Prinzip der Wissensstrukturierung, bei der die Objekte eines Gegenstandsbereichs (→Konzeptualisierung) durch binäre 'ist-ein'-Relationen zu einem hierarchischen Ordnungsschema angeordnet werden (S. 51).

Verhalten durch äußere Auswirkungen oder autonom hervorgerufene Änderungen von Prozesseigenschaften. Man unterscheidet dynamisches (transientes), stationäres und logisches Verhalten (S. 1).

Wechselwirkung (kooperative) bezeichnet den Sachverhalt, dass zwei Prozesse Einfluss aufeinander ausüben (S. 26). Dies äußert sich formal darin, dass einzelne Signale der Prozesse in Form einer kooperativen Wechselwirkungsrelation (S. 28, Gl. (2.67)) miteinander in Beziehung stehen (z.B. bijektive Abbildung bei I/O-Signalen, →Kausalität im Sinne der Kooperation).

Wechselwirkung (temporale) bezeichnet den Sachverhalt, dass der Verlauf eines Prozesses in der Vergangenheit Auswirkungen auf seinen zukünftigen Verlauf hat. Kennzeichen der temporalen Wechselwirkung von Prozessen ist das →Gedächtnis (S. 17).

Zustand nach [Waloschek 1998] die Form oder Art, in der ein Objekt (Körper, Substanz, System) in einem bestimmten Moment vorliegt (S. 1). In der Prozesstheorie heißen die →Signale eines →Markov-Prozesses Zustände (S. 32, Gl. (2.77)).

Literaturverzeichnis

[Ahrens 1997] Ahrens, W.; H.-J. Scheurlen; G.-U. Spohr (1997). *Informationsorientierte Leittechnik*. R. Oldenbourg Verlag.

[Ahrens 1994] Ahrens, W.; M. Polke (1994). *Informationsstrukturen in der Leittechnik*. In: Polke, M., Hrsg.: *Prozeßleittechnik*. R. Oldenbourg Verlag.

[Anderl 2002] Anderl, R.; S. Kleiner; M. Krastel (2002). *Produktdatenmanagement in der Simulation und Berechnung*. ProduktDatenJournal, 9(1):37–41.

[Arnold 1998] Arnold, M. (1998). *Kommunikationskonzept für die Prozessleittechnik*. Dissertation, Rheinisch-Westfälische Technische Hochschule Aachen, Lehrstuhl für Prozessleittechnik.

[v. Aspern 1994a] Aspern, Jens v. (1994a). *SPS-Softwareentwicklung - Petrinetze und Wortverarbeitung*. Hüthig Buch Verlag GmbH.

[v. Aspern 1994b] Aspern, Jens v. (1994b). *SPS-Softwareentwicklung mit Petrinetzen*. Hüthig Buch Verlag GmbH.

[Beaman 1988] Beaman, J.; R. Rosenberg (1988). *Constituive and modulation structure in bond graph modeling*. Journal of Dynamic Systems, Measurement and Control.

[Berard 2001] Berard, B.; M. Bidoit; A. Finkel (2001). *Systems and Software Verification*. Springer-Verlag.

[Berners-Lee 2001] Berners-Lee, T.; J. Hendler; O. Lassila (2001). *The Semantic Web*. http://www.scientificamerican.com.

[Bochmann 2000] Bochmann, D. (2000). *Zur Anwendung des Booleschen Differentialkalküls*. Vorlesungsmanuskript.

[Bochmann 1981] Bochmann, D.; C. Posthoff (1981). *Binäre dynamische Systeme*. Akademie Verlag Berlin.

[Booch 1991] Booch, G. (1991). *Object Oriented Design - with Applications*. The Benjamin/Cummings Publishing Company, Inc.

[Borutzky 2000] Borutzky, W. (2000). *Bondgraphen*. SCS-Europe BVBA.

[Broenink 1993] Broenink, J.; K. Wijbrans (1993). *Discribing discontinuities in bond graphs*. In: *Proceedings of the First International Conference On Bond Graph Modelling (ICBGM'93)*, S. 120–125. — SCS.

[Bronstein 1991] Bronstein, I. N.; K. A. Semendjajew (1991). *Taschenbuch der Mathematik*. Verlag Nauka, B. G. Teubner Verlagsgesellschaft, Verlag Harri Deutsch.

[Bronstein 1995a] Bronstein, I. N.; K. A. Semendjajew (1995a). *Teubner-Taschenbuch der Mathematik - Teil I*. B. G. Teubner Verlag.

[Bronstein 1995b] Bronstein, I. N.; K. A. Semendjajew (1995b). *Teubner-Taschenbuch der Mathematik - Teil II*. B. G. Teubner Verlag.

[Broy 2002] Broy, M.; J. Siedersleben (2002). *Objektorientierte Programmierung und Softwareentwicklung - Eine kritische Einschätzung*. Informatik Spektrum, 25:3–11.

[Buchner 1998] Buchner, H. (1998). *Grundlagen der nichtmetrischen Prozeßbeschreibung als Basis der informationsorientierten leittechnischen Modellbildung*. Dissertation, Rheinisch-Westfälische Technische Hochschule Aachen, Lehrstuhl für Prozessleittechnik.

[Burkhard 2003] Burkhard, H.-D. (2003). *Software-Agenten*. In: Görz, G., C.-R. Rollinger und J. Schneeberger, Hrsg.: *Handbuch der Künstlichen Intelligenz*, S. 943–1017. R. Oldenbourg Verlag.

[Chouikha 1999] Chouikha, M. (1999). *Entwurf diskret-kontinuierlicher Steuerungssysteme - Modellbildung, Analyse und Synthese mit hybriden Petrinetzen*. Dissertation, Technische Universität Braunschweig.

[Chouikha 1998] Chouikha, M.; B. Ober; E. Schnieder (1998). *Model-Based Control Synthesis for Discrete Event Systems*. In: Hamza, M., Hrsg.: *IASTED Conference on Modelling and Simulation*, S. 276–280. Pittsburgh (USA).

[Collatz 1949] Collatz, L. (1949). *Differentialgleichungen für Ingenieure*. Wissenschaftliche Verlagsanstalt Hannover, Wolfenbüttler Verlagsanstalt Wolfenbüttel.

[Cycorp] Cycorp. *The Syntax of CycL*. http://www.cyc.com/cycl.html.

[DARPA 2003] DARPA (2003). *The DARPA Agent Markup Language Homepage*. http://www.daml.org.

[Dauphin-Tanguy 1989] Dauphin-Tanguy, G.; C. Sueur; C. Rombaut (1989). *Bond-graph approach of commutating phenomena*. In: *Proceedings of Andvanced Information Processing in Automatic Control*. — IFAC.

[Decknatel 2001] Decknatel, G. (2001). *Entwicklung eines Typs kontinuierlich-diskreter höherer Petrinetze und seine Anwendung auf Bahnsysteme*. Dissertation, Technische Universität Braunschweig.

[Donges 2000] Donges, C.; M. Krastel (2000). *Das MechaSTEP-Projekt - ein Statusbericht*. ProduktDatenJournal, 7(1):13–14.

[Drath 2004] Drath, R.; Murat Fedai (2004). *CAEX - ein neutrales Datenaustauschformat für Anlagendaten - Teil 1*. atp - Automatisierungstechnische Praxis, 46(2):52 – 56.

[Däubler 1998] Däubler, L. (1998). *Produktdatentechnologie in derLeittechnik für den schienengebundenen Verkehr*. Diplomarbeit, Technische Universität Braunschweig, Institut für Regelungs- und Automatisierungstechnik.

[Däubler 2000a] Däubler, L.; E. Schnieder (2000a). *Distributed Engineering of Automation Systems with Hypernet*. In: Broecks, F. und L. Pauwels, Hrsg.: *Euromedia 2000*, S. 250–252. Antwerpen.

[Däubler 2000b] Däubler, L.; E. Schnieder (2000b). *Hypernet: a Distributed Engineering Tool for Collaborative Modeling with Petrinets*. In: al., U. F. Baake et, Hrsg.: *ECEC 2000, Concurrent Engineering in the Framework of its Convergence*, S. 183–186. Delft.

[Däubler 2004] Däubler, L.; U. Becker; R. Gratzke (2004). *Einsatzmöglichkeiten der Produktdatentechnologie in der Kfz-Funktionsentwicklung*. ProduktDaten-Journal, 11(1):46–49.

[Eaton 1997] Eaton, J. W. (1997). *GNU Octave A high-level interactive language for numerical computations*. www.octave.org/docs.html.

[Eibl et al. 2000] Eibl, M., D. Westphal und P. Hgorzelski (2000). *eCl@ss - ein Werkzeug zur Unterstützung des Prozesse im eCommerce, der Materialwirtschaft und der Analgendokumentation, bezogen auf das PLT-Gewerk*. atp - Automatisierungstechnische Praxis, 42(10):33–43.

[Engell 1997] Engell, S. (1997). *Modellierung und Analyse hybrider dynamischer Systeme*. at - Automatisierungstechnik, 45(4).

[Engell et al. 2002] Engell, S., G. Frehse und E. Schnieder, Hrsg. (2002). *Modelling, Analysis and Design of Hybrid Systems, LNCS 279*. Springer-Verlag.

[Fasol 1988] Fasol, K. H. (1988). *Binäre Steuerungstechnik*. Springer-Verlag.

[Feynberg 2001] Feynberg, I. (2001). *DAML+OIL Web Ontology Language*. http://www.w3.org/Submission/2001/12/.

[Gawthrop 1996] Gawthrop, P.; L. Smith (1996). *Metamodelling: Bond graphs and dynamic systems*. Prentice Hall.

[Gehrke 2001] Gehrke, J. (2001). *Untersuchung zum werkzeugunabhängigen Modelldatenaustausch in der Automatisierungstechnik*. Studienarbeit, Technische Universität Braunschweig, Institut für Regelungs- und Automatisierungstechnik.

[Genesereth] Genesereth, M. *Knowledge Interchange Format*. http://logic.stanford.edu/kif/kif.html.

[Genesereth 1989] Genesereth, M. R.; N. J. Nilsson (1989). *Logische Grundlagen der Künstlichen Intelligenz*. Vieweg Verlag.

[Ghezzi 1989] Ghezzi, C.; M. Jazayeri (1989). *Konzepte der Programmiersprachen*. R. Oldenbourg Verlag.

[GMA 2003] GMA, VDI/VDE-Gesellschaft Mess-und Automatisierungstechnik (2003). *VDI/VDE 3682 - Formalisierte Prozessbeschreibungen (Entwurf)*.

[Grose 2002] Grose, T.; G. Doney; S. Brodsky (2002). *Mastering XMI: Java Programming with XMI, XML, and UML*. John Wiley & Sons.

[Gruber 1995] Gruber, T. R. (1995). *Toward principles for the design of ontologies used for knowledge sharing*. International Journal of Human-Computer Studies, 43:907–928.

[Görz 2003] Görz, G.; C.-R. Rollinger; J. Schneeberger, Hrsg. (2003). *Handbuch der künstlichen Intelligenz*. Oldenbourg Wissenschaftsverlag GmbH.

[Guarino 1995a] Guarino, N. (1995a). *Formal ontology, conceptual analysis and knowledge representation*. International Journal of Human-Computer Studies, 43:625–640.

[Guarino 1998] Guarino, N. (1998). *Formal Ontology and Information Systems*. In: Guarino, N., Hrsg.: *Formal Ontology in Information Systems - Proceedings of the First International Conference FOIS'98*, S. 3–15. IOS Press.

[Guarino 1995b] Guarino, N.; P Giaretta (1995b). *Ontologies and Knowledge Bases*. In: Mars, N. J. I., Hrsg.: *Towards Very Large Knowledge Bases*. IOS Press.

[Hafner 2000] Hafner, C.; N. Noy (2000). *Ontological Foundations for Experimental Science Knowledge Bases*. Applied Artificial Intelligence, 14:565–618.

[Hanisch 1992] Hanisch, H.-M. (1992). *Petri-Netze in der Verfahrenstechnik*. R. Oldenbourg Verlag.

[Harel 1987] Harel, D. (1987). *Statecharts: a visual formalism for complex systems*. Science of Computer Programming, 9/1987:S. 231 – 274.

[Hegenbart 1984] Hegenbart, R. (1984). *Wörterbuch der Philosophie*. Humboldt Taschenbuchverlag.

[Holmevik 1995] Holmevik, J. R. (1995). *The History of SIMULA*. http://java.sun.com/people/jag/SimulaHistory.html.

[Horrocks et al.] Horrocks, I. et al. *The Ontology Inference Layer OIL*. http://www.ontoknowledge.org/oil/TR/oil.long.html.

[Horvitz 1992] Horvitz, E. J. (1992). *Automated Reasoning for Biology and Medicine*. Technischer Bericht, Knowledge Systems Laboratory, Stanford University.

[Isermann 1999] Isermann, R. (1999). *Mechatronische Systeme - Grundlagen*. Springer Verlag.

[ISO 1994a] ISO, International Standards Organisation (1994a). *ISO 10303-1: Overview und Fundamental Principles*.

[ISO 1994b] ISO, International Standards Organisation (1994b). *ISO 10303-11: Description methods: The EXPRESS language reference*.

[ISO 1996] ISO, International Standards Organisation (1996). *ISO-IEC/CD 10303-221:1996(E) Product data representation and exchange: Application protocoll: Electrotechnical design and installation*.

[ISO 1997] ISO, International Standards Organisation (1997). *ISO/CD 10303-221(E): Product data representation and exchange: Application protocol: Functional data and their schematic representation for process plant*.

[ISO 2001] ISO, International Standards Organisation (2001). *ISO/CD TS 10303-25: Product data representation and exchange: Implementation methods: EXPRESS to XMI Binding*.

[ISO 2002] ISO, International Standards Organisation (2002). *ISO/TS 10303-28: Product data representation and exchange: Implementation methods: XML representations fo EXPRESS schemas and data*.

[Jacobson 1998] Jacobson, Booch. G.; J. Rumbaugh; I. (1998). *The Unified Modeling Language User Guide*. Addison Wesley Longman Inc.

[Jacobson 1992] Jacobson, I. (1992). *Object-oriented software engineering : a use case driven approach*. Addison-Wesley Publishing Company.

[Jähnichen 2002] Jähnichen, S.; S. Herrmann (2002). *Was, bitte, bedeutet Objektorientierung?*. Informatik Spektrum, 25:266–276.

[Jiang 2005] Jiang, X. (2005). *Programming of a graphical tool for Bond Graph modeling and analysis*. Diplomarbeit, Technische Universität Braunschweig, Institut für Verkehrssicherheit und Automatisierungstechnik.

[Jörns 1995] Jörns, C.; L. Litz; S. Bergold (1995). *Automatische Erzeugung von SPS-Programmen auf der Basis von Petri-Netzen.* atp - Automatisierungstechnische Praxis, 37(3):11–14.

[Karnopp 1975] Karnopp, D.; R. Rosenberg (1975). *System Dynamics: A Unified Approach.* John Wiley and Sons.

[Klein 2000] Klein, M.; D. Fensel; F. van Harmelen; I. Horrocks (2000). *The Relation between Ontologies and Schema-languages: Translating OIL-specifications in XML Schema.* In: *Proceedings of the Workshop on Applications of Ontologies and Problem-solving Methods, 14th European Conference on Artificial Intelligence ECAI-00.*

[König 1988] König, R.; L. Quäck (1988). *Petri-Netze in der Steuerungs- und Digitaltechnik.* R. Oldenbourg Verlag.

[Kruchten 1999] Kruchten, P. (1999). *The Rational Unified Process.* Oldenbourg Wissenschaftsverlag GmbH.

[Lauber 1996] Lauber, J. (1996). *Methode zur funktionalen Beschreibung und Analyse von Produktionsprozessen als Basis zur Realisierung leittechnischer Lösungen.* Dissertation, Rheinisch-Westfff Technische Hochschule Aachen, Lehrstuhl für Prozessleittechnik.

[Ljung 1994] Ljung, L.; A. Glad (1994). *Modeling of dynamic systems.* Prentice-Hall International.

[Lubell 2002] Lubell, Joshua (2002). *From Model to Markup - XML Representation of Product Model Data.* In: N., N., Hrsg.: *XML 2002 Conference Proceedings.* www.idealliance.org/papers/xml02.

[Lunze 2002a] Lunze, J. (2002a). *Regelungstechnik 2.* Springer-Verlag.

[Lunze 2002b] Lunze, J. (2002b). *What is a hybrid system?.* In: Engell, S., G. Frehse und E. Schnieder, Hrsg.: *Modelling, Analysis and Design of Hybrid Systems, LNCS 279.* Springer-Verlag.

[Meixner 2004] Meixner, U. (2004). *Einführung in die Ontologie.* Wissenschaftliche Buchgesellschaft.

[Merz 2000] Merz, R.; L. Litz (2000). *Objektorientierte mathematische Modellierung.* Informatik Spektrum, 23(2).

[Mintert 2003] Mintert, S. (2003). *Abgehoben - das Semantische Web.* iX, 6/2003:S. 90–92.

[Mircecsu 1997] Mircecsu, A. (1997). *Über die Beschreibung und Optimierung verteilter Automatisierungssysteme.* Dissertation, TU Braunschweig.

[Müller 1997] Müller, K. (1997). *Entwurf robuster Regelungen*. B. G. Teubner Verlag.

[Modelica Association 2000] Modelica Association (2000). *Modelica - A Unified Object-Oriented Language for Physical Systems Modeling - Tutorial*. www.modelica.org/documents/ModelicaTutorial14.pdf.

[Morgeneyer 2003] Morgeneyer, T.; D. Püschmann (2003). *Modellbildung und Simulation hybrider Systeme mit kombinierter Bondgraphen/Petrinetz-Beschreibung*. Diplomarbeit, Technische Universität Braunschweig, Institut für Verkehrssicherheit und Automatisierungstechnik.

[Mosterman 1997] Mosterman, P. J. (1997). *Hybrid Dynamic Systems: A hybrid Bond Graph Modeling paradigm and its application in diagnostics*. Dissertation, Nashville University.

[Nebot 1999] Nebot, A.; F. Cellier; F. Mugica (1999). *Simulation of Heat and Humidity Budgets of Biosphere 2 Without Air Conditioning*. Ecological Engineering, 13:333–356.

[Nenninger 2002] Nenninger, G.; G. Frehse; V. Krebs (2002). *Reachability Analysis and Control of a Special Class of Hybrid Systems*. In: Engell, S., G. Frehse und E. Schnieder, Hrsg.: *Modelling, Analysis and Design of Hybrid Systems, LNCS 279*. Springer-Verlag.

[Nenninger 2001] Nenninger, G. M. (2001). *Modellbildung und Analyse hybrider dynamischer Syteme als Grundlage für den Entwurf hybrider Steuerungen*. Dissertation, Universtität Karlsruhe.

[Ober 1999] Ober, B. (1999). *Modellgestützte Synthese ereignisdiskreter Steuerungen*. Dissertation, Technische Universität Braunschweig, Institut für Regelungs- und Automatisierungstechnik.

[Oestereich 2001] Oestereich, B. (2001). *Objektorientierte Softwareentwicklung - Analyse und Design mit der Unified Modeling Language*. Oldenbourg Wissenschaftsverlag GmbH.

[OMG 2002] OMG (2002). *Meta Object Facility (MOF) Specification, Version 1.4*. http://www.omg.org/docs/formal/02-04-03.pdf.

[OMG 2003a] OMG (2003a). *OMG Unified Modeling Language Specification, Version 1.5*. http://www.omg.org/docs/formal/03-03-01.pdf.

[OMG 2003b] OMG (2003b). *XML Metadata Interchange (XMI) Specification, Version 2.0*. http://www.omg.org/docs/formal/03-05-02.pdf.

[Ontoprise] Ontoprise. *OntoEdit Manual*. Download unter www.ontoprise.de.

[Ortner 2000a] Ortner, E. (2000a). *Wissensmanagement - Teil1: Rekonstruktion des Anwendungswissens*. Informatik Spektrum, 23(2).

[Ortner 2000b] Ortner, E. (2000b). *Wissensmanagement - Teil2: Systeme und Werkzeuge*. Informatik Spektrum, 23(3).

[Pallaske 1994] Pallaske, U. (1994). *Das Wissen über den Prozeß - Prozeßmodelle*. In: Polke, M., Hrsg.: *Prozeßleittechnik*. R. Oldenbourg Verlag.

[Partsch 1998] Partsch, H. (1998). *Requirements-Engineering systematsich*. Springer-Verlag.

[Paynter 1961] Paynter, H. M. (1961). *Analysis and Design of Engineering Systems*. The MIT Press (vergriffen).

[Petri 1962] Petri, C. A. (1962). *Kommunikation mit Automaten*. Dissertation, Universität Bonn.

[Polke 2003] Polke, B.; E. Schnieder (2003). *Formalisierte Prozessbeschreibungen - Entwurf der Richtlinie VDI/VDE 3682 und deren Anwendung*. atp - Automatisierungstechnische Praxis, (8).

[Polke 1993] Polke, M. (1993). *Einführung in die Prozessleittechnik*. Vorlesung, Lehrstuhl für Prozeßleittechnik, RWTH Aachen.

[Polke 1994] Polke, M. (1994). *Einleitung*. In: Polke, M., Hrsg.: *Prozeßleittechnik*. R. Oldenbourg Verlag.

[Polke 2000] Polke, M. (2000). *Gedanken zum 'Informationsmodell'*. Technischer Bericht, GMA Fachausschuß 7.21.

[Puppe et al. 2003] Puppe, F., H. Stoyan und R. Studer (2003). *Knowledge Engineering*. In: Görz, G., C.-R. Rollinger und J. Schneeberger, Hrsg.: *Handbuch der Künstlichen Intelligenz*, S. 599–641. R. Oldenbourg Verlag.

[Rath 1998] Rath, W., Hrsg. (1998). *Aristoteles - Die Kategorien - Griechisch/Deutsch*. Philipp Reclam jun.

[Reinhardt 1991] Reinhardt, F.; H. Soeder (1991). *dtv-Atlas zur Mathematik - Band I*. dtv - Deutscher Taschenbuchverlag.

[Reisig 1985] Reisig, W. (1985). *Systementwurf mit Netzen*. Springer-Verlag.

[Reisig 1986] Reisig, W. (1986). *Petrinetze - Eine Einführung*. Springer-Verlag.

[Rumbaugh et al. 1993] Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy und W. Lorensen (1993). *Objektorientiertes Modellieren und Entwerfen*. Coedition Carl Hanser Verlag / Prentice-Hall.

[Schneider 1997] Schneider, H.-J. et al. (1997). *Lexikon Informatik und Datenverarbeitung, Version 4.0*. Oldenbourg Verlag.

[Schnieder 1993] Schnieder, E. (1993). *Prozessinformatik*. Vieweg Verlag.

[Schnieder 2002] Schnieder, E. (2002). *Regelungstechnik I*. Vorlesungsmitschrift, TU Braunschweig.

[Schnieder 2003] Schnieder, E. (2003). *Integration heterogener Modellwelten der Automatisierungstechnik*. In: M. Nagel, B. Westfechtel, Hrsg.: *Modelle, Werkzeuge und Infrastrukturen zur Unterstützung von Entwicklungsprozessen*. WILEY-VCH.

[Schnieder 2001] Schnieder, E.; L. Jansen (2001). *Begriffsmodelle der Automatisierungstechnik - Basis effizienten Engineerings*. In: Schnieder, E., Hrsg.: *EKA 2001 Engineering komplexer Automatisierungssysteme, 7. Fachtagung*, S. 1–27. Braunschweig.

[Staab 2002] Staab, S. (2002). *Wissensmanagement mit Ontologien und Metadaten*. Informatik Spektrum, 25(3):194–209.

[Strömberg 1994] Strömberg, J.-E. (1994). *A mode switching modelling philosophy*. Dissertation, Linsköpings Universitet.

[The Math Works 1994] The Math Works (1994). *MATLAB Technical Report: Algebraic Loops and S-functions*. Technischer Bericht, The Math Works.

[Unbehauen 1997] Unbehauen, R. (1997). *Systemtheorie 1*. R. Oldenbourg Verlag.

[Voss 2001] Voss, S.; K. Gutenschwager (2001). *Informationsmanagement*. Springer Verlag.

[W3C 2000] W3C (2000). *XML Schema*. http://www.w3c.org/XML/Schema.

[W3C 2003a] W3C, Semantic Web Activity (2003a). *OWL Web Ontology Language Guide*. http://www.w3.org/TR/owl-guide/.

[W3C 2003b] W3C, Semantic Web Activity (2003b). *Resource Description Framework*. http://www.w3c.org/RDF/.

[W3C 2003c] W3C, Semantic Web Activity (2003c). *Web-Ontology (WebOnt) Working Group*. http://www.w3.org/2001/sw/WebOnt/.

[Waloschek 1998] Waloschek, P. (1998). *Wörterbuch der Physik*. dtv - Deutscher Taschenbuchverlag.

[Wegner 1987] Wegner, P. (1987). *Dimensions of object-based language design*. In: *Proceedings of OOPSLA'87*.

[Willke 1998] Willke, H.; D. Gnewekow (1998). *Systemisches Wissensmanagement*. Verlag Lucius & Lucius.

[Witt 1992] Witt, K.-U. (1992). *Einführung in die objektorientierte Programmierung*. R. Oldenbourg Verlag.

[Wunsch 1985] Wunsch, G. (1985). *Geschichte der Systemtheorie*. R. Oldenbourg Verlag.

[Wunsch 2000] Wunsch, G. (2000). *Grundlagen der Prozesstheorie*. B. G. Teubner Verlag.